`SERVOUS`: Cross-Technology Neighbour Discovery and Rendezvous for Low-Power Wireless Devices

Rainer Hofmann
Institute of Technical Informatics
Graz University of Technology, AT
rainer.hofmann@tugraz.at

Carlo Alberto Boano
Institute of Technical Informatics
Graz University of Technology, AT
cboano@tugraz.at

Kay Römer
Institute of Technical Informatics
Graz University of Technology, AT
roemer@tugraz.at

Abstract

Cross-technology communication (CTC) supports a direct message exchange between different wireless technologies, enabling explicit interaction between devices with incompatible physical layer (PHY). State-of-the-art work typically neglects the integration of CTC alongside the native communication stack of a device. As result, current schemes assume that CTC can be carried out at any time and that devices know their neighbours in advance, which is unrealistic or impractical for most duty-cycled low-power wireless systems. In this paper, we fill this gap and present `SERVOUS`, a cross-technology neighbour discovery and rendezvous protocol allowing a device to autonomously discover and communicate with surrounding nodes operating on another PHY, while still operating at low duty cycle. `SERVOUS` reuses a fraction of the radio idle time to discover the presence of nearby appliances capable of CTC and learns their configuration. It then exploits the Chinese remainder theorem to determine the minimum amount of idle time that needs to be reused to guarantee a cross-technology rendezvous, and can adjust itself to maximize energy efficiency while satisfying specific application requirements. We base `SERVOUS` on low-power probing to minimize the channel utilization and integrate it into IEEE 802.15.4 and Bluetooth Low Energy devices running Contiki. We further evaluate the performance of `SERVOUS` experimentally and show its ability to let heterogeneous low-power wireless devices interact with each other using CTC, without affecting their native communications.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Design, Algorithms, Performance, Efficiency.

Keywords

Cross-Technology Communication, Internet of Things, Media Access Control, Neighbour Discovery, Rendezvous.

1 Introduction

Cross-technology communication (CTC) has emerged as a suitable alternative to multi-radio gateways to allow a direct interaction between wireless devices with incompatible physical layer (PHY). CTC is usually achieved by making use of a mutually-available side channel and allows the creation of attractive services, such as clock synchronization [1], channel coordination [2], and sensor re-configuration [3]. A large body of work has focused on devices sharing the 2.4 GHz ISM band and designed CTC schemes enabling a direct communication between popular wireless technologies such as Bluetooth Low Energy (BLE), IEEE 802.15.4, and Wi-Fi [4]. In particular, existing CTC approaches make use of either packet-level modulation or PHY emulation to convey information across devices using different standards. In *packet-level modulation*, data can be encoded into different frame lengths [5], gap durations [6], beacon intervals [7], or power levels [8]; 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 frame is adjusted such that a portion of it can be recognized by a device using another technology as a legitimate frame, thereby achieving a high throughput [9–12].

State-of-the-art work on CTC has mainly focused on demonstrating that a data exchange across various technologies is possible and on achieving a high throughput or long range [11, 13]. However, they often rely on strong assumptions (e.g., that all devices know about each other's existence beforehand) and have ignored the challenge of *integrating CTC with the native communication stack of a device* (i.e., they assume that a device can carry out CTC at anytime).

CTC is not intended as a standalone functionality. Cross-technology communication is commonly meant as an add-on feature giving a device the ability to interact with nearby appliances *alongside its normal operations*. Therefore, CTC functionality (i.e., the transmission and reception of cross-technology frames) should coexist in parallel and not interfere with the normal communications of a device. This is especially challenging for low-power wireless systems, which make use of duty-cycled MAC protocols to orchestrate communication while minimizing their energy consumption. Indeed, radio duty-cycling operations cannot be easily postponed or rearranged on an individual device, as this would affect the connectivity with the other nodes in the network. Hence, one *cannot allocate the radio for CTC activities* arbitrarily or permanently, as currently done in most CTC prototypes [6–11]. Moreover, the time a device spends car-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2021
17–19 February, Delft, The Netherlands © 2021 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-5-2

rying out CTC should be kept low to affect the energy budget of a device only minimally – an aspect that has rarely been considered by existing work [14]. To ensure a seamless coexistence alongside existing communications, the transmission and reception of cross-technology frames should hence take place only *when a device's radio is idle*. Even better, CTC activities should occur during a tiny portion of this idle time, so to minimize energy expenditure.

Ensuring a cross-technology rendezvous. The reuse of a radio's idle time, however, makes it challenging to sustain a successful and efficient cross-technology data exchange, as devices employ different standards and are unaware of each other's radio activities. As a result, a device may be transmitting cross-technology frames during a portion of idle time that does not overlap with the period during which its intended receiver listens for incoming CTC transmissions. To enable and guarantee a *cross-technology rendezvous*, i.e., the existence of a common timeslot during which two devices with incompatible PHY can complete a cross-technology data exchange, several key challenges need to be tackled.

First, every device should be able to *model its radio activity* and exchange this information with devices in its surroundings, so to inform them about its idle time and hence about the chances of successfully establishing a cross-technology rendezvous. Such radio activity model needs to be *generic*, i.e., applicable to different wireless standards, as well as *lightweight*, i.e., a device should be able to share this information by sending only a few bytes over the air.

Second, exchanging information with surrounding devices entails the ability to discover their presence and to initiate an interaction despite the incompatible PHY. There is hence a need for a *cross-technology neighbour discovery* scheme that allows to find surrounding devices employing different wireless standards. Unfortunately, existing neighbour discovery schemes focus only on devices using the same technology, requiring the same duty cycle [15–20] or foregoing extensive pre-configuration [21–24]. This calls for novel neighbour discovery approaches, which should be agnostic to the technology employed by the communicating devices – a complex task, as many CTC schemes require prior knowledge about the involved devices or only allow a unidirectional exchange between two specific technologies.

Third, all cross-technology data exchanges need to be *highly efficient* to let a device operate at low duty cycle at all times. A device should hence be able to anticipate when its intended peers are available to carry out CTC (so to minimize its radio on-time), as well as to initiate a data transmission only if the receiver's radio is known to remain in an idle state for a sufficient time to complete the data exchange.

Our contributions. In this paper we present `SERVOUS`, a cross-technology neighbour discovery and rendezvous protocol allowing low-power wireless devices using incompatible PHYs to autonomously find and directly communicate with each other, while still operating at low duty cycle. In particular, `SERVOUS`[1] reuses a portion of the time during which a device's radio is idle to exchange cross-technology frames with surrounding appliances, while ensuring a seamless coexistence with existing MAC protocols. To this end, each device builds a lightweight model capturing when and how often its radio is active or in low-power mode. By sharing this model with nearby devices during a discovery phase and by using the Chinese remainder theorem (CRT) [25], `SERVOUS` determines the minimum amount of time α during which a device should keep its radio active to *guarantee* a cross-technology rendezvous. `SERVOUS` can further adjust α to minimize the radio-on time, while still maximizing the probability of rendezvous and while satisfying specific application requirements on the acceptable latency of the cross-technology rendezvous or on the increased duty cycle.

`SERVOUS` neither requires time synchronization nor assumes prior knowledge about the communicating devices, other than the existence of a common RF channel and CTC alphabet. The exchange of cross-technology frames and the discovery procedure are both *receiver-initiated* using a scheme resembling low-power probing [26]: this allows to minimize the radio-on time and the RF spectrum utilization.

We further integrate `SERVOUS` into several *off-the-shelf* BLE and IEEE 802.15.4 devices running Contiki, making CTC functionality a practical add-on feature alongside the normal operations of a device. Our implementation can be installed besides the native communication stack of a device and does not require any modification to its duty-cycling strategy, preventing compatibility problems.

Finally, we evaluate `SERVOUS` experimentally, quantifying its performance and memory footprint, as well as showcasing that it enables CTC between heterogeneous low-power wireless devices without affecting normal communications and without significantly increasing their energy expenditure.

In summary, this paper makes the following contributions:

- We present `SERVOUS`, a cross-technology neighbour discovery and rendezvous protocol for low-power wireless devices, highlighting its key design principles (Sect. 2);
- We show how to model and exchange the radio activity of low-power wireless devices, which allows `SERVOUS` to operate only when their radio is idle (Sect. 3);
- We illustrate in detail the design of `SERVOUS`, describing how it carries out discovery and establishes rendezvous using a receiver-initiated scheme (Sect. 4);
- We show how `SERVOUS` computes the minimum amount of time in which a device should keep its radio active to guarantee a cross-technology rendezvous, and how it can help achieving a low energy consumption (Sect. 5);
- We describe `SERVOUS`' seamless integration into the Contiki operating system and its implementation on off-the-shelf BLE and IEEE 802.15.4 devices (Sect. 6);
- We evaluate the performance of `SERVOUS` experimentally and showcase its operations (Sect. 7).

After discussing `SERVOUS`' limitations and how to further increase its energy efficiency and throughput by making devices aware of their hardware characteristics (Sect. 8), we review related work (Sect. 9) and conclude the paper (Sect. 10).

2 SERVOUS: Design Rationale

We motivate our discussion by describing a scenario in which two co-located low-power wireless devices employing

[1] *Servus* is an informal greeting / salutation used in many parts of Central and Eastern Europe that originates from Latin and means "at your service".

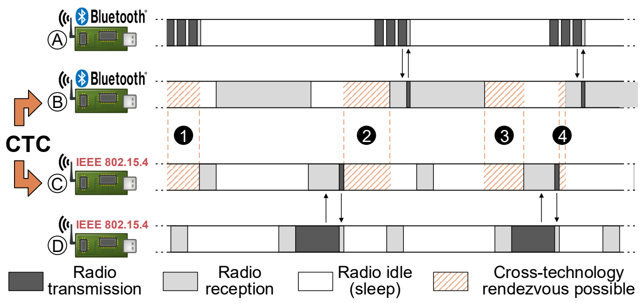


Figure 1: Sketch of the problem tackled by SERVOUS: two devices Ⓑ and Ⓒ using different low-power wireless technologies want to interact using CTC besides their normal operations. Due to the existing communication activities, a cross-technology rendezvous is only possible at irregular intervals (marked in orange). Without prior knowledge of their configuration, the two devices should still find each other and exchange cross-technology frames in a bounded time, while reusing as little of their radio's idle time as possible.

diverse technologies operating in the same frequency band (e.g., BLE and IEEE 802.15.4) want to exchange data using CTC *besides* their normal operations. This can be useful, for example, to coordinate the channel usage and improve coexistence [2], or to synchronize the device clocks, thereby enabling a consistent time-stamping of related events [1,27].

Such a scenario is depicted in Fig. 1. Devices Ⓐ and Ⓑ form a BLE network, with Ⓐ sporadically broadcasting advertisements and Ⓑ periodically scanning for them. Devices Ⓒ and Ⓓ form an IEEE 802.15.4 network and make use of a low-power listening protocol such as ContikiMAC [15], where devices periodically perform a clear channel assessment (CCA) to assess whether waking up to receive a message (and confirm its reception) or keep sleeping.

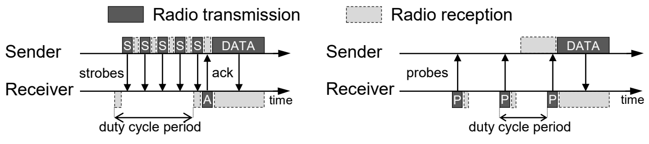
Let's suppose that device Ⓑ and Ⓒ want to interact using CTC. As they do not want to affect their ongoing communications, the two devices should exchange cross-technology frames only when their radio is idle. Due to the different medium access control scheme and duty cycle configuration, the instants of time in which the radio of both devices are idle are scattered and irregular [28]. Therefore, unless the two devices have been pre-configured with a very detailed knowledge of each other's protocol and configuration, neither Ⓑ nor Ⓒ knows beforehand *when* and for *how long* its counterpart can carry out CTC, i.e., when a *cross-technology rendezvous* is possible (in the case of Fig. 1, during the four time-slots coloured in orange). On the one hand, Ⓑ and Ⓒ could simply attempt a cross-technology data transmission or reception *whenever* their radio is idle. This, however, would result in a very large energy expenditure draining the devices' battery, especially if Ⓑ and Ⓒ need to exchange data frequently. On the other hand, the two devices could attempt a cross-technology data transmission or reception only occasionally during their idle time, hence preserving their limited energy budget. When doing this blindly, however, there is no guarantee of a successful rendezvous, which is undesirable, as it leads to potentially unbounded delays when the two devices attempt a cross-technology data exchange.

Lightweight model of the radio usage. With SERVOUS, we tackle this problem by firstly getting a detailed understanding of when and for how long a device's radio is idle, so to schedule CTC activities without affecting normal operations. To this end, we exploit the fact that the majority of low-power wireless systems have a well-defined periodic access to the media. In the example shown in Fig. 1, one can indeed observe how both Ⓑ's scanning interval and Ⓒ's periodic CCA check follow a periodic pattern, which can be modelled as described in Sect. 3. We show that such a periodic access to the media can be inferred based on the knowledge of the employed MAC protocol and its parameters, and later learnt from an analysis of the radio access timings. This allows SERVOUS to make use of a lightweight model in which two parameters are sufficient to describe the behaviour of a device's radio and schedule CTC activities accordingly. Moreover, by exchanging this model with nearby devices during a cross-technology discovery, SERVOUS can guarantee a cross-technology rendezvous in bounded time, even when reusing only a tiny portion of a device radio's idle time.

Cross-technology broadcast transmissions. To keep our solution generic and allow communication between arbitrary technologies sharing the same frequency, we let SERVOUS exchange cross-technology frames using packet-level modulation. In contrast to CTC solutions making use of PHY emulation (which can sustain a very high throughput, but are often limited to one direction only [9–12]), this allows cross-technology *broadcast* transmissions, i.e., the dispatching of cross-technology frames to multiple devices using diverse technologies at once[2]. This does not only simplify discovery significantly (as a device can advertise its presence to multiple recipients at once), but further enables a *bidirectional* data exchange allowing, for example, the transmission of acknowledgement messages confirming a successful discovery or rendezvous. SERVOUS can hence be implemented on top of packet-level modulation CTC schemes such as X-Burst [14], which was shown to enable a cross-technology broadcast communication among *off-the-shelf* Wi-Fi, BLE and IEEE 802.15.4 devices [29]. Such CTC schemes, however, encode information into the length of data packets and decode it by performing a high frequency sampling of the received signal strength, which results in long transmission times (e.g., ≈ 63 ms for a 20-byte frame). Because of this, the design of SERVOUS needs to be steered in the direction of minimizing the amount of data exchanged via CTC, also to avoid an unnecessary congestion of the RF channel.

Spectrum-friendly receiver-initiated scheme. We hence base SERVOUS on a receiver-initiated scheme in which a device periodically sends short cross-technology probes to announce that it is awake and ready to receive a cross-technology message through high-frequency RSS sampling. This resembles the operations of low-power probing (LPP) protocols [26], where a device willing to transmit data to one of its neighbours listens for the corresponding probe and sends information upon its reception. In contrast to sender-

[2]Using packet-level modulation, one can also exchange data between devices using the same wireless standard, which enables communication between co-located networks using the same technology.



(a) low-power listening (b) low-power probing

Figure 2: Sender- vs. receiver-initiated approaches.

initiated schemes such as low-power listening (LPL) [17], where a transmitter repeatedly sends data (e.g., strobes [18]) until the intended recipient signals its reception, receiver-initiated schemes require less information to be sent, as shown in Fig. 2. This result in a lower channel utilization and a higher spectrum friendliness, which is important given that devices share the same frequency channel to carry out CTC. The use of a receiver-initiated scheme also allows a device to discover and keep track about the presence of nearby appliances by passively listening to probes, as discussed in Sect. 4.

Energy-efficient rendezvous in bounded time. Traditional LPL and LPP protocols are based on a key assumption, namely that sender and receiver operate on the same duty cycle. This way, a sender can be sure to have a rendezvous with its intended receiver by sending strobes (LPL) or by listening for probes (LPP) for an entire duty cycle period. However, as CTC is an add-on feature carried out when a device's radio is idle, sender and receiver do not have a common notion of duty cycle period, as shown in Fig. 1. A fundamental design challenge in SERVOUS is hence how often and for how long a sender should listen for probes to successfully establish a rendezvous with the intended receiver. By sharing the model describing the behaviour of a device's radio during discovery, SERVOUS is able to calculate the *probability* as well as an *upper bound on the latency* of a successful cross-technology rendezvous, which are both proportional to the amount of the radio's idle time α used to listen for probes. Based on specific application requirements (e.g., on the acceptable latency of the cross-technology rendezvous or on the acceptable increase in duty cycle when enabling CTC), SERVOUS can then adjust α to minimize the necessary radio on-time for a successful rendezvous, thereby reducing energy consumption.

3 Modelling the Radio Activities of a Device

To schedule CTC activities without affecting normal operations, SERVOUS needs to model the usage of a device's radio and reuse portions of its idle time. This is possible, as the majority of low-power wireless systems uses duty-cycled MAC protocols to schedule communications: this results in a well-defined periodic access to the media that can be exploited to create a lightweight model describing the radio activities of a device. Specifically, one can model a radio as either performing its *usual activities* (classical communications) or as *idle* (low-power mode), as illustrated in Fig. 3.

Note that the grey area in Fig. 3 marked as "usual activities" does not necessarily imply that the radio is continuously active the entire time, but could also represent a sequence of radio on/off states, as discussed below. This

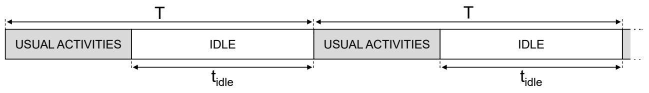


Figure 3: Lightweight model of low-power wireless devices.

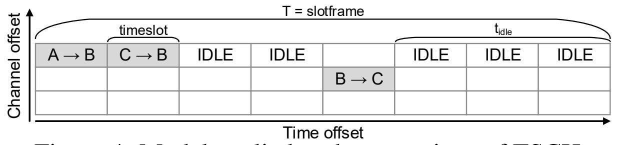


Figure 4: Model applied to the operations of TSCH.

simplification allows to identify periodic patterns more easily and let SERVOUS seamlessly schedule CTC during t_{idle}. Moreover, by exchanging the wake-up interval T during a cross-technology discovery, SERVOUS can guarantee an efficient cross-technology rendezvous, as discussed in Sect. 4.

We show next how this radio activity model with only two parameters can be applied to devices using popular low-power wireless technologies like BLE and IEEE 802.15.4.

3.1 IEEE 802.15.4

MAC protocols based on IEEE 802.15.4 can be synchronous or asynchronous. We discuss in detail Time Slotted Channel Hopping (TSCH) and ContikiMAC as an example of synchronous and asynchronous protocol, respectively.

Time Slotted Channel Hopping. TSCH is a MAC layer specified in the IEEE 802.15.4-2015 amendment that builds a globally synchronized mesh network. Essentially, time is sliced into periodically repeating *timeslots* that are grouped into one or more *slotframes*, as shown in Fig. 4. Each timeslot can be allocated to one or more pairs of devices for their communications. In case a timeslot is not allocated, the radio remains idle during this time. Therefore, one can identify a period T corresponding to the slotframe length, and t_{idle} as the longest idle duration (i.e., the longest number of consecutive timeslots during which the radio is idle). In the example shown in Fig. 4, the first five slots of the slotframe would be considered as usual activities, and the last three as idle time: a portion of the latter may be exploited to carry out CTC. Note that aggregating assigned and idle timeslots (e.g., the first 5 timeslots) reduces the amount of idle time that can be reused by SERVOUS. This is not an issue, as CTC is an add-on feature and our aim is ensuring a seamless coexistence with normal operations while guaranteeing a cross-technology data exchange rather than achieving a high CTC throughput.

Whilst the slotframe size of a TSCH network is typically static and fixed at compile time, the timeslots are often allocated dynamically. For this reason, T does not change at runtime, but t_{idle} may vary: it is hence important that either the new schedule is communicated to SERVOUS, or that the latter adjusts the model autonomously, for example by analysing the radio access timings, as discussed in Sect. 3.3.

ContikiMAC. Developed by Dunkels [15], ContikiMAC is an asynchronous sender-initiated protocol based on LPL, whose operations are illustrated in Fig. 5. Nodes periodically wake up every t_{wake_up}, perform a clear channel assessment (t_{cca}), and remain active to receive messages (t_{rx}) if they detect activity on the channel. Otherwise, they return to low-power mode until the next scheduled wake-up time.

When carrying out broadcast transmissions, a device sends a message repeatedly for the full duration of a wake-up interval ($t_{tx\_broadcast}$) to ensure that all neighbours are able to receive it. When carrying out unicast transmissions, a device follows the same principle, but can stop sending data as soon

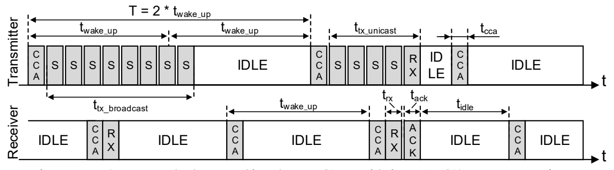


Figure 5: Model applied to ContikiMAC's operations.

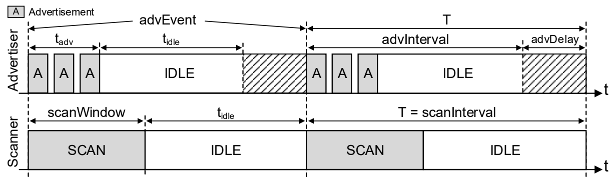


Figure 6: Model applied to connection-less BLE.

as a link-layer ACK is received ($t_{tx\_unicast}$). For our model, we need to account that a device can act as both transmitter and receiver over time. To derive t_{idle}, we consider that a receiver may be idle for at most t_{wake_up} - t_{cca} - t_{rx} - t_{ack}, where t_{rx} corresponds to the necessary time to receive a frame with the maximum length allowed by the standard (127 bytes). To derive T, we consider that a transmitter may use up to one full wake-up interval to send data during a broadcast transmission, and that during the following t_{wake_up} remains idle (because $t_{tx\_broadcast}$ is slightly larger than t_{wake_up} to ensure that all neighbours can be reached, causing the device to omit the CCA check): from this, we derive that $T = 2 \cdot t_{wake\_up}$.

3.2 Bluetooth Low Energy

BLE devices can operate in two different modes: *connection-less* and *connection-based*.

Connection-less mode. When using connection-less mode, a BLE device can either operate as *advertiser* or *scanner*. Advertisers periodically transmit a message (on up to the three advertising channels) at the beginning of each advertising event (*advEvent*), which has a fixed duration specified by the *advInterval* parameter plus a pseudo-random offset specified by the *advDelay* parameter (uniformly distributed between 0 and 10 ms). Scanners, instead, periodically listen for messages (for a duration specified by the *scanWindow* parameter) at the beginning of each *scanInterval*, as shown in Fig. 6. One can hence model T as the scanner's *scanInterval* and the advertiser's *advInterval* + 5 ms[3]. Instead, t_{idle} is proportional to the difference between *scanInterval* and *scanWindow* for a scanner. For an advertiser, t_{idle} is proportional to the difference between *advInterval* and the duration of three consecutive advertisement messages (t_{adv}), which is bounded to 30 ms by the BLE specification [30].

Connection-based mode. When using connection-based mode, BLE devices operate in a periodic manner, acting either as *master* or *slave*, as depicted in Fig. 7. After a connection is established, devices periodically exchange data at the beginning of connection events (*connEvent*), as specified by the *connInterval* parameter. Master and slave can exchange multiple frames within one connection event, up to a maximum connection time (*connMaxTime*): one can hence subtract this value from the *connInterval* to derive a

[3]In SERVOUS, we account for the randomness introduced by *advDelay* by anticipating or postponing the transmission of a probe, as shown in Sect. 6.

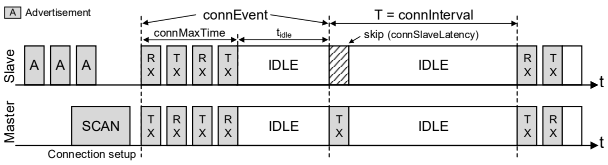


Figure 7: Model applied to connection-based BLE.

minimum t_{idle} for the slave. As the master can have several slaves, t_{idle} corresponds to the largest (*connInterval* − *connMaxTime*) across all slaves. The period T corresponds to the *connInterval* for the slave and to the sum of all the connection intervals of the connected slaves for the master[4].

The parameters of a BLE connection can be changed at runtime [31]: as for TSCH, it is hence important that either the new parameters are communicated to SERVOUS, or that the latter adjusts the model autonomously, for example by analysing the radio access timings, as discussed in Sect. 3.3.

3.3 Deriving the Model

As just discussed, the radio activity model used by SERVOUS can be derived using knowledge of the employed MAC protocol and its parameters. To this end, SERVOUS can be configured at compile time accordingly. However, in case a protocol configuration is changed at runtime (e.g., whenever the BLE connection's parameters or the TSCH schedule has changed), it is necessary to update the model. Whilst the protocol could directly convey this information to SERVOUS using an application programming interface, SERVOUS can also learn the new radio settings autonomously by analysing the *function calls* to the radio, i.e., by deriving a sequence of "*radio on*" and "*radio off*" durations [14]. If the operations (and peculiarities) of a protocol are known, it is relatively simple to derive T and t_{idle} from such a sequence. For example, when using ContikiMAC, one can look for the two consecutive CCA checks carried out within t_{cca} at the beginning of each wake-up interval to infer the beginning of T. Similarly, on a BLE scanner, every "*radio on*" call corresponds to the beginning of a *scanWindow*. In our current implementation, we follow this approach to learn the two model parameters based on knowledge of the employed protocol (see Sect. 6). In principle, one could also learn the model parameters without such knowledge by reusing existing work on periodicity detection in time series [32–34]: this is an add-on to SERVOUS that is beyond the scope of this paper.

4 SERVOUS: Working Principle

To let a device signal its presence and availability for CTC, SERVOUS periodically broadcasts a *probe* (solid red rectangle) at the beginning of each idle phase, and listens for a reply (dashed light blue rectangle), as seen in Fig. 8.

A probe consists only of a short address (ID) that allows to identify a device: this is important to minimize the amount of transmitted data and hence the energy consumption, as the transmission of 1-byte of data using packet-level CTC takes roughly 2-3 ms [14]. Upon a probe reception, a device can trigger the transmission of either a cross-technology discov-

[4]To improve energy efficiency, a BLE slave can skip a number of connection events, as specified by the *connSlaveLatency* parameter (in the example shown in Fig. 7, the second *connEvent* is skipped). In our model, we are assuming *connSlaveLatency* = 0 as we need to account for the worst case.

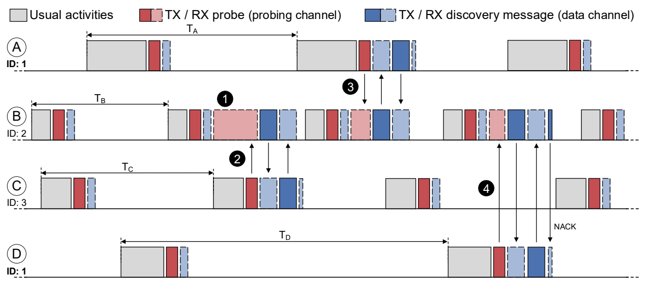


Figure 8: Cross-technology neighbour discovery in `SERVOUS`.

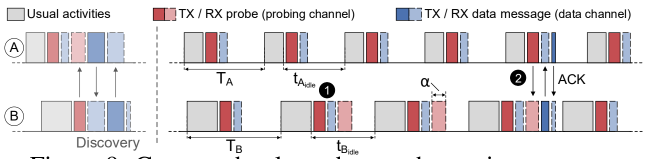


Figure 9: Cross-technology data exchange in `SERVOUS`.

ery frame (if the device is unknown) or a CTC frame containing the data to be exchanged (if the device is known).

Cross-technology neighbour discovery. To discover nearby devices or double-check their reachability, it is hence sufficient to reuse a portion of the radio idle time to listen for probes (dashed light red rectangle) and transmit a *discovery request* to which a device answers with a *discovery reply* (solid blue rectangles). These two messages contain a mapping between the short address used for probing and the MAC/IPv6 address of the device, as well as the two model parameters T and t_{idle}, which can be used to guarantee rendezvous in bounded time (as detailed in Sect. 5) and to infer the length of the CTC frame to be transmitted, respectively.

In the example shown in Fig. 8, Ⓑ wants to discover devices in its surroundings and hence starts the discovery procedure by listening for probes during its radio idle time ❶. Every device periodically signals its presence by broadcasting probes: in this case ❷, the probe of Ⓒ is the first to be received by Ⓑ, who will answer with a unicast *discovery request* communicating its short address (ID: 2), its MAC/IPv6 address, as well as its model parameters T_B and $t_{B_{idle}}$. Device Ⓒ updates its neighbour table and answers with a unicast *discovery reply* containing its short address (ID: 3), its MAC/IPv6 address, T_C, and $t_{C_{idle}}$. This way, also Ⓑ updates its neighbour table and the two devices have mutually discovered each other (or confirmed their reachability). The same procedure is followed upon reception of Ⓐ's probe ❸.

Fig. 8 also shows an example of discovery without a positive ending: in this case, Ⓓ is a hidden terminal to Ⓐ (and vice-versa), but both devices can talk to Ⓑ. In this case, Ⓐ and Ⓓ have the same short address (ID 1), which is derived as a hash function of their MAC/IPv6 address. As Ⓑ already has one entry in its neighbour table with ID 1, it will answer to Ⓓ's *discovery reply* with a NACK message indicating the duplicate short address in its neighbour table ❹. Upon reception of this message, Ⓓ should change its ID and perform a new discovery, during which it will automatically inform all its neighbours about the change of its short address when sending out *discovery request* messages. Note that a device receiving a probe with the same ID of its own will immediately change its short address and trigger a new discovery.

This way, `SERVOUS` ensures that there are no duplicate addresses within a two-hop neighbourhood, which allows to cope with hidden terminals and ensures that *short* probes can be used, thereby ensuring energy efficiency. The size of the short address can be determined at compile time depending on the number of expected nearby devices and on the chosen hash function's ability to minimize the number of collisions.

The device discovery procedure terminates as soon as no new device is detected within a given time. The procedure is triggered at device startup as well as periodically: the interval can be chosen at compile time based on the expected dynamicity of the network, but can also be adapted at runtime if several devices have joined the network or are no longer reachable with respect to the previous discovery. Note that a device still broadcasts probes while in discovery mode: this maximizes the chances that two devices carrying out discovery at the same time find each other.

Cross-technology data exchange. Once two devices have discovered each other, they can exchange cross-technology data. Also this process follows a receiver-initiated approach: a device willing to transmit data using CTC listens for the probe of the intended device, which indicates its availability to receive a CTC frame. Differently from the discovery procedure, a device looks for the intended probe using *only a portion* of its idle duration (α), which allows to trade the probability and latency of a successful rendezvous for energy efficiency, as discussed in Sect. 5. `SERVOUS` also makes use of *two separate RF channels* for probing and for data exchange (discovery requests and replies also use the data channel), both known at compile time by all devices: this allows to minimize collisions and channel congestion.

Fig. 9 illustrates the data exchange between device Ⓐ and Ⓑ, who have previously successfully discovered. Device Ⓑ wants to transmit data to Ⓐ: after its usual activities, Ⓑ first sends a probe to advertise its presence during its idle time and shortly listens for replies ❶. If no node is trying to contact Ⓑ, the device starts listening for Ⓐ's probe for a portion of its idle time α. This operation is repeated until a probe from device Ⓐ is received: thereafter, Ⓑ can send its data and Ⓐ can optionally acknowledge its reception ❷. When sending its data, Ⓑ can infer the length of the CTC frame that can be received by Ⓐ from the $t_{A_{idle}}$ parameter of the model exchanged during discovery. Once Ⓑ has no more data to send, it stops listening for Ⓐ's probes and only keeps broadcasting probes periodically, remaining idle otherwise.

5 Efficient Rendezvous in Bounded Time

The duration of α plays a key role w.r.t. the probability of successful rendezvous, its potential maximum latency (Ω), as well as the necessary radio on-time until its completion. `SERVOUS` can pick the most efficient α that guarantees (or maximizes the probability of) a successful rendezvous based on application-specific requirements, as we show next.

5.1 Probability of Finding a Probe

In our context, rendezvous occurs when the transmission of a probe falls within the time during which a device listens for it: the latter corresponds to α, as discussed in Sect. 4.

To *bound* the rendezvous' latency as a function of α and to find the minimum length of α that *guarantees* a rendezvous in bounded time (α_{min}), we divide time into *discrete*

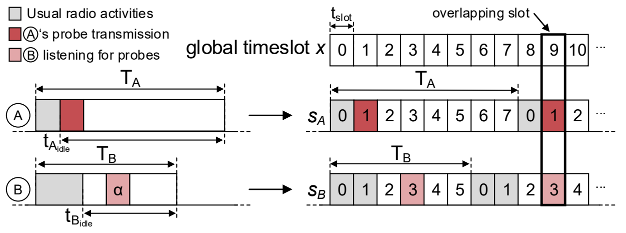


Figure 10: Illustration of the rendezvous problem: Ⓐ sends a probe in its second slot within T_A, whereas Ⓑ listens for probes in its fourth slot within T_B: the two overlap at $x = 9$. Our aim is to predict and bound the time until an overlap.

timeslots with fixed length t_{slot}, as shown in Fig. 10. We assume that a probe is transmitted in the first idle slot after the usual activities[5], and that a device willing to transmit data listens for the recipient's probes for a time α right afterwards.

Fig. 10 exemplifies the rendezvous problem with two devices Ⓐ and Ⓑ having $T_A = 8$ and $T_B = 6$. In this example, Ⓑ wants to transmit data to Ⓐ and is hence looking for its probes. Assuming a global timebase x, Ⓐ's probe is detected by Ⓑ at global timeslot $x = 9$, i.e., when Ⓐ's second timeslot within T_A and Ⓑ's fourth timeslot within T_B overlap.

In order to calculate such an overlapping timeslot among two devices, the global timeslot x needs to be mapped to the timeslot s of each device. As devices are operating periodically, this can be done using the modulo operator:

$$x \equiv s \pmod{m} \qquad \text{with} \qquad m = {}^{T}\!/_{t_{slot}} \tag{1}$$

where T is the period and m the amount of timeslots within T. Thus, finding the global timeslot x in which two specific timeslots of Ⓐ and Ⓑ overlap can be described as a pair of congruences $x \equiv s_A \ (mod\ m_A)$ and $x \equiv s_B \ (mod\ m_B)$.

Co-prime periods. Such a set of congruences is known to always have a solution by the Chinese remainder theorem when m_A and m_B are *relatively prime* (co-prime), i.e., when their greatest common divisor (gcd) is 1 [25]. Thus, if

$$\gcd(m_A, m_B) = 1, \tag{2}$$

there exists a solution for every possible combination of timeslots among both devices, i.e., *each* timeslot of Ⓐ overlaps with *each* timeslot of Ⓑ, within their *common period* $T_{AB} = T_A \cdot T_B$. As an example, consider two devices Ⓐ and Ⓑ with $T_A = 40$ ms and $T_B = 50$ ms. We define $t_{slot} = 10$ ms, which results in $m_A = 4$ and $m_B = 5$, respectively.

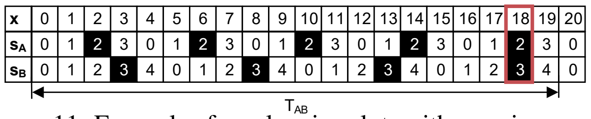
As $gcd(4,5) = 1$, it is guaranteed that each timeslot of Ⓐ will overlap with each timeslot of Ⓑ within their common period $T_{AB} = 200$ ms (20 timeslots). Fig. 11 shows the resulting timeslot distribution until T_{AB}: Ⓐ's third timeslot and Ⓑ's fourth timeslot overlap at $x = 18$, i.e., within the 20th timeslot. Indeed, the CRT gives as solution $x = 18$ for the two congruences $x \equiv 2 \ (mod\ 4)$ and $x \equiv 3 \ (mod\ 5)$.

Non co-prime periods. All previous observations hold only when m_A and m_B are co-prime. If this is not true, namely:

$$\gcd(m_A, m_B) = g \qquad \text{with} \qquad g \neq 1, \tag{3}$$

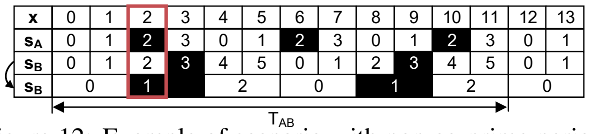
there still exists an overlapping timeslot every g timeslots: $s_A \equiv s_B \ (mod\ g)$, i.e., *one* timeslot of Ⓐ overlaps with exactly

[5]After a probe transmission, a device also checks for incoming answers to the probe. Note that, in principle, any idle slot can be selected for the probe transmission, as long as this choice is consistent over time.



x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_A	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0
s_B	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0

Figure 11: Example of overlapping slots with co-prime periods.



x	0	1	2	3	4	5	6	7	8	9	10	11	12	13
s_A	0	1	2	3	0	1	2	3	0	1	2	3	0	1
s_B	0	1	2	3	4	5	0	1	2	3	4	5	0	1
s_B	0		1		2		0		1		2		0	

Figure 12: Example of scenario with non co-prime periods. When combining timeslot into a multislot, one can ensure that each of A's timeslots overlaps with each of B's multislots.

one out of g consecutive timeslots of Ⓑ within their common period T_{AB}. In the following, such a set of g consecutive timeslots is referred to as *multislot*. Thus, by listening for a multislot, Ⓑ is able to find the probe of Ⓐ even if m_A and m_B are not relatively prime. As $gcd > 1$, T_{AB} is defined by:

$$T_{AB} = {}^{T_A \cdot T_B}\!/_{\gcd(T_A, T_B)}. \tag{4}$$

As an example, consider two devices Ⓐ and Ⓑ with $T_A = 40$ ms and $T_B = 60$ ms, as shown in Fig. 12. We keep $t_{slot} = 10$ ms as in the previous example, which results in $m_A = 4$ and $m_B = 6$. Ⓐ's third timeslot and Ⓑ's fourth timeslot will never overlap, i.e., a rendezvous cannot be established. However, as $g = gcd(4,6) = 2$, one can define a multislot as two consecutive timeslots: when doing so, one can guarantee again rendezvous. Suppose that Ⓐ sends probes in timeslot 2 (γ) and that Ⓑ listens for them in every multislot 1. By satisfying the equation $\gamma \equiv s_B \ (mod\ 2)$, one can compute which timeslots of Ⓑ overlap with γ. For $s_B = 0$ (multislot #0), $s_B = 2$ (multislot #1) and $s_B = 4$ (multislot #2), using the CRT, one can derive $x = 6$, $x = 2$, and $x = 10$. In the example shown in Fig. 12, Ⓑ looks for probes in its second multislot, which corresponds to $x = 2$.

Based on the above, the minimum listening time α_{min} that ensures the detection of a probe can hence be defined as:

$$\alpha_{min} = \gcd(T_A, T_B). \tag{5}$$

Note that there are cases where a device's maximum idle time (t_{idle}) available to listen for probes is shorter than the required α_{min}, e.g., when both devices operate on the same wake-up interval or are a multiple of each other. In those cases, a device would have to listen for the whole wake-up interval ($\alpha_{min} = T$) to ensure a rendezvous: this, however, would defeat SERVOUS' purpose, as one would affect the usual communications of the device. Although a guarantee cannot be provided in such cases, one can still calculate the probability of a successful rendezvous as:

$$P = {}^{t_{idle}}\!/_{\gcd(T_A, T_B)} \tag{6}$$

5.2 Upper Bound for Rendezvous

In Sect. 5.1, the upper bound Ω on the latency of a rendezvous was defined as the common period T_{AB} of two devices. However, when listening for probes for longer than α_{min}, one can significantly reduce Ω. To compute Ω as a function of an arbitrary α value, one can use Algorithm 1, which takes as input the period of the two devices (T_A and T_B), as well as the duration of a timeslot (t_{slot}). As discussed in Sect. 4, this algorithm is carried out after two devices have discovered and exchanged their model parameters.

Algorithm 1: Upper bound for rendezvous.

```
   Input: $T_A$, $T_B$, $\alpha$, $t_{slot}$
   Output: $\Omega$
1  $m_A = T_A/t_{slot}$, $m_B = T_B/t_{slot}$, $n_B = \alpha/t_{slot}$
2  $s_{total} = m_A$, $s_{checked} = []$, $T = 0$, $flag = false$
3  if $\alpha \geq T_A$ then
4      return $T_A$
5  end
6  for $i = 0$; $i < m_A$; $i++$ do
7      $flag = false$
8      for $j = 0$; $j < n_B$; $j++$ do
9          $slot = (i * m_B + j) \; mod \; m_A$
10         if $slot$ not in $s_{checked}$ then
11             add $slot$ to $s_{checked}$
12             $s_{total} --$
13             $flag = true$
14         end
15     end
16     if $s_{total} == 0$ then
17         return $(n_B + m_B * T) * t_{slot}$
18     else if $flag == false$ then
19         return $(n_B + m_B * (T-1)) * t_{slot}$
20     end
21     T++
22 end
```

The algorithm first translates the two periods as well as α into discrete timeslots of length t_{slot}. It then verifies if α, used by device Ⓑ to detect probes from Ⓐ, is greater or equal than T_A: if this is the case, then Ⓑ can surely detect the probe within T_A. If $\alpha < T_A$, the algorithm determines the maximum amount of periods during which Ⓑ has to look for probes to ensure that one is surely be detected. Due to the periodicity of T_A and T_B, there are exactly m_A periods of Ⓑ that are of interest. For each of them, the timeslots used by Ⓑ to listen for probes are mapped to a timeslot of Ⓐ (line 9). The algorithm checks if there was already an overlap with this timeslot (line 10); if not, the amount of possible timeslots of Ⓐ is reduced. The algorithm terminates if the amount of possible timeslots drops to zero, or when no new timeslot of Ⓐ was found within a period of Ⓑ.

5.3 Misaligned Timeslots and Clock Drift

So far, we have assumed that the timeslots of the two devices are fully aligned. In reality, this assumption does not hold, as heterogeneous devices run independently, are unsynchronized, and may experience a clock drift over time.

Misaligned timeslots. Fig. 13 shows the implication of misaligned timeslots: the rendezvous is no longer at $x = 18$ (as in Fig. 11), but at $x = 14$. Indeed, for a rendezvous to be successful, it is fundamental that the beginning of the probe (i.e., the beginning of Ⓐ's third timeslot) is contained within Ⓑ's (multi)slot. This is no longer the case at $x = 18$: instead, the probe is detected at $x = 14$, which may force Ⓑ to use also a portion of its fifth timeslot to listen for Ⓐ's probe. Still, Ⓐ's probe transmission will overlap with one (multi)slot of Ⓑ within the common period T_{AB}, i.e., the rendezvous will only be shifted within T_{AB} and the computation of Ω is unaffected.

Clock drift. The clock drift, instead, has a more severe impact, as it may affect the periodicity of devices. Indeed, as shown in Fig. 14, the clock drift causes slots to move away, potentially causing the absence of an alignment at $x = 18$ (i.e., a rendezvous may not be established even when listening for probes for α_{min}). However, the drift of interest for the establishment of a rendezvous is limited to T_{AB} and is inde-

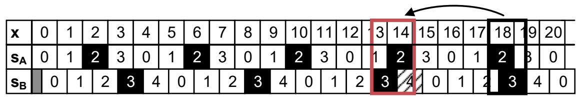


Figure 13: Rendezvous are shifted due to misaligned slots.

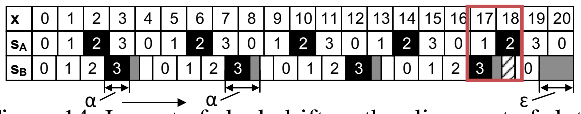


Figure 14: Impact of clock drift on the alignment of slots.

pendent across data exchanges. Hence, the maximum clock drift ε between two devices within T_{AB} can be determined as:

$$\varepsilon = \frac{T_A \cdot T_B}{\gcd(T_A, T_B)} \cdot 2 \cdot \delta, \tag{7}$$

where δ is the worst-case drift of a device expressed in ppm. A survey of available `HC-49S` packaged crystals [35] reveals that the majority of inexpensive parts experiences a drift over the temperature range [-20°C, $+70$°C] of $\delta = \pm 50$ ppm.

To compensate for the drift ε, the minimum listening time α_{min} needs to be adjusted accordingly:

$$\alpha_{min} = \begin{cases} \frac{T_A \cdot T_B \cdot 2 \cdot \delta}{\gcd(T_A, T_B)}, & \text{if } \varepsilon > \gcd(T_A, T_B) \\ \gcd(T_A, T_B), & \text{otherwise} \end{cases} \tag{8}$$

In particular, to ensure overlapping slots, and thus the establishment of rendezvous, α needs to be greater than the maximum drift within the common period of both devices. Hence, the drift can be neglected over short timescales, but needs to be compensated for longer ones using Eq. 8.

5.4 Energy-efficient Rendezvous

As discussed in Sect. 5.2, by listening for probes for longer than α_{min}, a device can significantly reduce Ω. However, different values of α result not only in a different Ω, but also in a different radio-on time R_{ON}, which has a direct implication on the energy consumption of the device.

Fig. 15 shows the worst-case R_{ON} and the upper bound on the rendezvous latency Ω among two devices Ⓐ and Ⓑ ($T_A = 250$ ms, $T_B = 197$ ms, $T_{A_{idle}} = 117$ ms, $T_{B_{idle}} = 186$ ms) for different values of α in range $[0, T_B]$. Specifically, Fig. 15 shows that, whilst Ω decreases as α increases, the resulting worst-case R_{ON} follows an irregular pattern as α changes.

As its ultimate goal is to enable CTC communication on top of existing activities without significantly increasing the energy expenditure, `SERVOUS` allows to choose α such that a minimum R_{ON} is sustained while satisfying specific application requirements. To this end, `SERVOUS` first computes the suitable range of values for α. As discussed in Sect. 5.3, α_{min} can be computed using Eq. 8, which results in 5 ms when assuming a drift $\delta = \pm 50$ ppm. The maximum length of α (α_{max}) is limited by the remaining idle duration of Ⓐ and Ⓑ, as well as by the amount of data that needs to be exchanged between the two devices. In this example, we assume that Ⓑ wants to send Ⓐ 10 bytes of data without expecting an ACK, which results in a $\alpha_{max} = 148$ ms (the transmission and reception of a probe takes 8 ms in our implementation, whereas transmitting a 10 bytes payload takes 35 ms).

After deriving α_{min} and α_{max}, `SERVOUS` loops through all the values within this range and determines for each of them the worst-case rendezvous latency Ω using Algorithm 1, as well as the worst-case radio-on time as $R_{ON} = \alpha \cdot \Omega/T_B$.

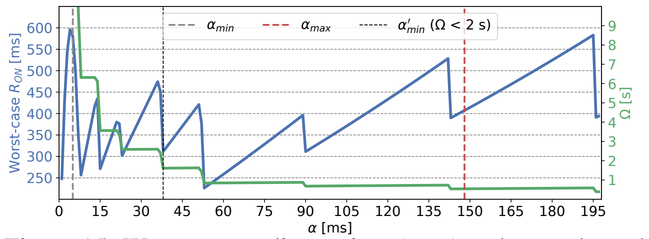


Figure 15: Worst-case radio-on time (R_{ON}) and upper bound on the rendezvous latency (Ω) as a function of α's duration.

Accounting for application-specific requirements. Depending on the application's requirements, SERVOUS can pick the most suitable value of α. For example, if the latency of the rendezvous is irrelevant, SERVOUS can simply use the α leading to the lowest R_{ON} within $[\alpha_{min}, \alpha_{max}]$. In the example shown in Fig. 15, this corresponds to $\alpha = 53$ ms.

If the latency of the rendezvous matters, α_{min} will be adapted accordingly. Specifically, SERVOUS selects as new α'_{min} the first value that allows to sustain less than the desired Ω (in Fig. 15, $\Omega < 2$ s) and picks the α value leading to the lowest R_{ON} within $[\alpha'_{min}, \alpha_{max}]$. An application may also want to privilege energy-efficiency and specify, for example, that SERVOUS can only increase a device's radio-on time by 10 %. In this case, SERVOUS adjusts α_{max} accordingly as $\alpha'_{max} = DC_{inc} \cdot T$, where DC_{inc} is the maximum percentage increase in radio-on time within T. SERVOUS then picks the α value leading to the lowest R_{ON} within $[\alpha_{min}, \alpha'_{max}]$.

6 Implementation

We implement SERVOUS in Contiki on top of X-Burst [14]. The latter is a packet-level CTC framework allowing to encode data into the duration of energy bursts (generated by transmitting legitimate packets), and to decode information by means of high-frequency RSS sampling. As X-Burst can already determine ContikiMAC's wake-up interval at runtime, we extend its functionality to let SERVOUS model t_{idle} and the radio activities of connection-less BLE devices as described in Sect. 3. To compensate for the randomness introduced by the *advDelay*, we anticipate or postpone the transmission of a probe. Specifically, we use a *compensation window*, limited by t_{idle} and by the maximum data to be exchanged (i.e., two discovery messages); initially transmitting the CTC probe in the middle of this window. If, over time, the transmission of a CTC probe falls outside this compensation window (because of too many consecutive anticipations or posticipations), SERVOUS would either affect the usual behaviour of a device or reduce the maximum amount of CTC data that can be exchanged. To avoid this, we postpone the probe transmission so that it can be sent again at the middle of the next compensation window: this ensures that the usual behaviour of a device is unaffected, although it breaks SERVOUS' periodic operations[6]. The transmission of probes and data packets is made on two separate channels located at 2450 and 2460 MHz, respectively. The transmission of CTC data is carried out using X-Burst's default alphabet, which describes the set of properties used to convey information via CTC, e.g., the mapping between symbols and burst durations. The CTC frames contain a preamble consisting of five bursts of pre-defined duration, a header specifying the type of message and other options (e.g., whether an ACK is requested), as well as a 1-byte checksum to verify integrity. Each device has its own *neighbour table*, containing a 8-bit short ID, the 64-bit MAC address, as well as T, t_{idle}, α, and Ω for each neighbour. We use discrete timeslots with $t_{slot} = 1$ ms: this allows to control α in a fine-grained way.

[6]Alternatively, one can adjust the start of the next *advEvent*: this allows to retain SERVOUS' periodicity while affecting a device's usual behaviour.

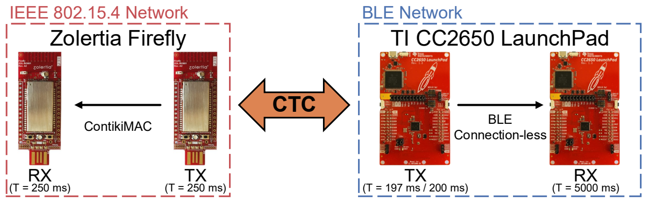


Figure 16: Experimental setup used to evaluate SERVOUS.

Hardware platforms. We implement SERVOUS on two *off-the-shelf* IoT devices supported by Contiki, namely the TI CC2650 LaunchPad supporting both, BLE and IEEE 802.15.4, as well as the Zolertia Firefly employing an IEEE 802.15.4-compatible TI CC2538 transceiver.

7 Evaluation

We evaluate SERVOUS experimentally. We first showcase its functionality by setting up two independent heterogeneous networks and let two devices carrying out CTC in parallel to their normal communications, i.e., resembling the setting depicted in Fig. 1 (Sect. 7.1). Thereafter, we evaluate SERVOUS' memory footprint (Sect. 7.2), discovery latency (Sect. 7.3), and energy consumption (Sect. 7.4).

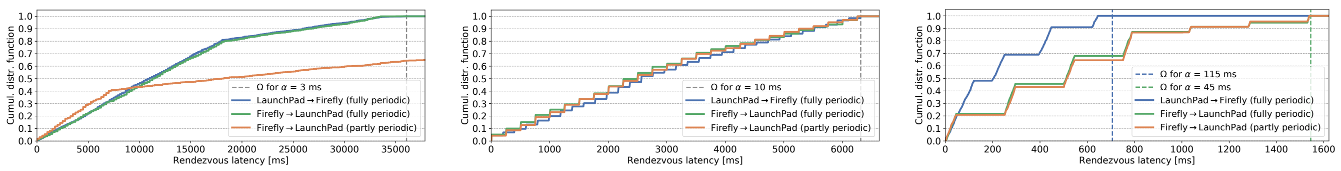
Experimental setup. We set up two networks as shown in Fig. 16. Two Zolertia Firefly act as an IEEE 802.15.4-based network using ContikiMAC (with its default channel check rate of 8 Hz), and two TI CC2650 LaunchPads act as a BLE-based network[7]. One of the CC2650 LaunchPads acts as BLE advertiser with an *advInterval* of 192 or 195 ms (depending on the evaluation type), resulting in $T = 197$ or 200 ms (including the 5ms advDelay) and $t_{idle} = 186$ or 189 ms. The other CC2650 LaunchPad acts as a BLE scanner with a *scanInterval* of 5000 ms and a *scanWindow* of 2000 ms (default scan parameters on Android: SCAN_MODE_BALANCED). This results in $T = 5000$ ms and $t_{idle} = 3000$ ms. For the Zolertia Firefly devices, only each second idle phase is used by SERVOUS resulting in $T = 250$ ms and $t_{idle} = 117$ ms, as explained in Sect. 3.1.

We carry out all experiments in an RF interference-free environment with the devices placed at 1 m distance and using a transmission power of 0 dBm. During all evaluations, SERVOUS runs alongside a device's normal activities.

7.1 Upper Bounds on Rendezvous' Latency

We start by checking whether SERVOUS can guarantee rendezvous in bounded time as illustrated in Sect. 5 and whether the calculated upper bounds on the rendezvous latency Ω hold. To this end, we measure the time between a device starts listening for a probe and the probe being successfully detected. We distinguish between two cases: co-prime and

[7]The TI CC2650 LaunchPad can operate either as BLE or IEEE 802.15.4 device: in all our experiments, we use it as BLE device only.



(a) $\alpha = 3$ ms (minimum) (b) $\alpha = 10$ ms (common value) (c) $\alpha = 45$ ms / 115 ms (maximum)

Figure 17: Measured rendezvous latency when using co-prime periods ($T = 197$ ms on the LaunchPad and 250 ms on the Firefly).

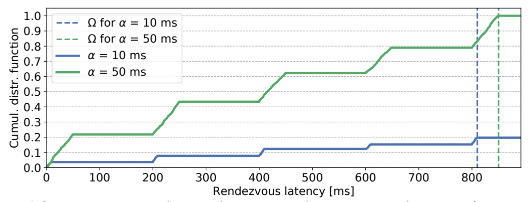


Figure 18: Measured rendezvous latency when using non co-prime numbers ($T = 200$ ms on the TI CC2650 LaunchPad and 250 ms on the Zolertia Firefly for different values of α).

non co-prime wake-up intervals T. Each experiment is repeated 1000 times with the devices being restarted, in order to generate different constellations of slot alignments between the two devices. We also run SERVOUS on different pairs of devices to show the universality of the approach.

Co-prime periods. We run SERVOUS on both transmitting devices and set the *advInterval* of BLE to 192 ms, which makes the two periods co-prime, as $gcd(250, 197)$=1. Fig. 17 shows the cumulative distribution function (CDF) over the measured rendezvous latency of all 1000 runs for different α values and for different directions (BLE $\rightarrow$ IEEE 802.15.4, IEEE 802.15.4 $\rightarrow$ BLE). The dashed lines indicate the upper bound Ω calculated by SERVOUS. When using $\delta = 50$ ppm, we derive $\alpha_{min} = 5$ ms, although we have noticed that the drift is actually much lower than that (we could correctly operate even at $\alpha_{min} = 3$ ms). α_{max} is specified as 45 and 115 ms on the Zolertia Firefly and the TI CC2650 LaunchPad, respectively, which allows the exchange of 20 bytes of data. When making use of larger values of α (Fig. 17(b) and (c)), the duration of a rendezvous never exceeds the upper bound Ω computed by SERVOUS, confirming the correctness of its estimation. For smaller values of α, the upper bound may not be met if SERVOUS' operations are not fully periodic (partly periodic – orange line). As discussed in Sect. 6, to compensate the *advDelay* randomness, we shift the probe within a 115 ms window, which may cause SERVOUS to postpone the transmission of some probes. If, instead, we retain SERVOUS' periodicity by adjusting the start of the next *advEvent* whenever the transmission of a CTC probe falls outside the compensation window, the latency of a rendezvous never exceeds the upper bound Ω (fully periodic – green / blue line).

Non co-prime periods. We change the *advInterval* of the TI CC2650 LaunchPad from 192 to 195 ms and let SERVOUS run on the receiving Zolertia Firefly. This makes the wake-up intervals no longer co-prime as $gcd(250, 200) = 50$. Thus, as explained in Sect. 5.1, α_{min} needs to be at least 50 ms to guarantee rendezvous. Fig. 18 shows the measured latency of 1000 rendezvous for different values of α. When using $\alpha = 50$ ms, all rendezvous occur within $\Omega = 850$ ms, as computed by SERVOUS. When using $\alpha = 10$ ms, the probability of rendezvous within 810 ms drops to 20 % as shown by Eq. 6.

Table 1: Memory footprint for different hardware platforms.

Hardware platform	RAM / ROM (kB)		
	w/o SERVOUS	w/ SERVOUS	**SERVOUS only**
LaunchPad	16.48 / 50.57	18.36 / 59.90	**1.88 / 9.33**
Firefly	5.86 / 20.41	7.60 / 30.16	**1.74 / 9.75**

Coexistence with other communications. During the previous evaluation, both networks were operating normally while SERVOUS was running in parallel. Specifically, the two Zolertia Firefly have exchanged one message every second and the BLE advertiser transmitted a message every 197 or 200 ms (depending on the setup). To verify that SERVOUS runs seamlessly on top of existing devices, we measured the packet reception rate (PRR) of all communications, including the CTC exchange of 20 bytes on every rendezvous. The PRR has been 100% for all our experiments, showing that SERVOUS does indeed not affect existing communications.

7.2 Memory Footprint

We quantify next the memory footprint of SERVOUS using the `gcc-size` command. Table 1 shows the memory footprint of SERVOUS in terms of RAM and ROM usage for both, the TI CC2650 LaunchPad and the Zolertia Firefly. The table also reports the memory footprint of the entire Contiki application running with and without SERVOUS. The tiny differences in SERVOUS' memory footprint between the two platforms are due to slightly different implementations of X-Burst, on top of which SERVOUS is built. With a footprint below 2 kB of RAM and 10 kB of ROM for both platforms, SERVOUS is well-suited for resource-constrained IoT devices.

7.3 Discovery Latency

We evaluate next how long it takes to discover nearby devices as a function of their number. To this end, we use one Zolertia Firefly and five TI CC2650 LaunchPads (with the same wake-up interval). We measure how much time it takes for a Zolertia Firefly (with $\alpha = 45$ ms) to successfully discover up to five surrounding TI CC2650 LaunchPads that are manually configured with non-duplicated addresses. As for Sect. 7.1, all devices are randomly restarted 1000 times in order to generate different constellations of slot alignments. Fig. 19 shows the time it takes to discover the surrounding devices as a function of their number. As expected, as all devices have the same wake-up period T, the discovery latency is proportional to the upper bound on the necessary time to discover a probe times the number of nearby devices. The high number of outliers are due to the random restart of all devices and their rather small and similar wake up intervals.

7.4 Power Consumption

We evaluate next the additional power consumption introduced by SERVOUS. To this end, we measure the average power consumption of the devices used in our evaluation setup using a Monsoon power monitor. Specifically, we first

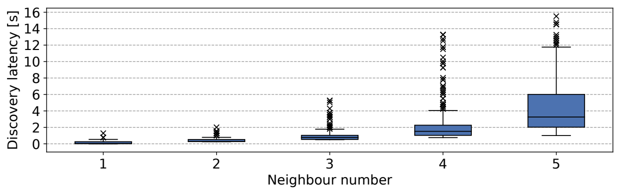


Figure 19: Discovery latency for the 1st, 2nd, 3rd, 4th, and 5th neighbour independent of the actually discovered device.

run all devices without SERVOUS to have a baseline. We then repeat the same measurements when SERVOUS is running on the devices. As the power consumption strongly depends on the use case, i.e., on how long and how often a device is discovering, on how frequently rendezvous are established, as well as on the relation between the devices' wake-up intervals, we measured the average power consumption while a device is probing only. Table 2 shows the results: SERVOUS' probing message is relatively lightweight and increases the power consumption on the Zolertia Firefly by only 12 and 21% when in transmitting and receiving mode, respectively. On the TI CC2650 LaunchPad, the power consumption increases minimally on the BLE scanner (0.47%) and by 112% of the BLE advertiser. However, please note that the BLE advertiser's power consumption is in the order of sub-mW: therefore, despite this difference SERVOUS is quite efficient.

8 Discussion, Limitations, and Future Work

Broadcast support. SERVOUS is designed and optimized for unicast communication. A device willing to transmit to all nearby devices the same data should hence send individual unicast messages to all entries in its neighbour table.

Phase lock. Instead of continuously listening for a probe, the transmitter could estimate the recipient's next wake-up time and use a form of phase lock (e.g., as in ContikiMAC [15]). This can decrease the necessary radio-on time for rendezvous even further: we will investigate this in future work.

Increasing CTC throughput. The throughput of packet-level CTC approaches strongly depends on the hardware characteristics of the involved devices, e.g., on the frequency of RSS sampling, and on the speed at which radio instructions can be loaded and executed [14]. Based on knowledge of those hardware characteristics, one can derive a faster alphabet (i.e., the set of properties used to encode symbols) to increase the throughput between a group of devices. For example, X-Burst embeds a dedicated module that aids the automated creation of a CTC alphabet that is supported by two or more communicating devices [14]. SERVOUS could hence allow an exchange of these parameters during discovery and store in the neighbour table which alphabet could be used to speed-up a cross-technology data exchange. We will implement this functionality in future work.

Support of other technologies. Our implementation focuses on BLE and IEEE 802.15.4 devices given their pervasiveness in today's IoT landscape. However, in principle, SERVOUS is generic and can be used with other technologies operating in the 2.4 GHz band supported by X-Burst [29]. When it comes to Wi-Fi devices, which also use these frequencies, one should note that these devices typically do not sleep: one should hence explicitly allocate a portion of their time to be devoted for CTC (t_{idle}) on a periodic basis (T).

Table 2: Power consumption with and without SERVOUS.

Device	T (ms)	Avg. power cons. (mW)	
		w/o SERVOUS	w/ SERVOUS
LaunchPad (TX)	197	0.84	1.87
LaunchPad (RX)	5000	10.7	10.75
Firefly (TX)	250	23.37	26.18
Firefly (RX)	250	10.8	13.16

Impact of RF interference and collisions. The computation of the probability of successful rendezvous and of the upper bound on its latency described in Sect. 5 currently assumes absence of collisions when devices broadcast a cross-technology probe. While the use of a dedicated channel and the short probe duration helps in this regard, collisions may occur or the presence of RF interference may prevent the correct probe reception on one or more nearby devices. With knowledge of the channel occupancy or of the link's packet reception rate, one could revise the equations to account for this probability and adjust the upper bound accordingly. We will implement this functionality in future work.

9 Related Work

We analyse next related work on CTC as well as on device discovery and rendezvous in low-power wireless networks.

Cross-technology communication. A large body of work has explored how to enable communication between devices with incompatible PHY. Chebrolu et al. [5] have been among the first to highlight the possibility of exploiting a side channel to enable CTC. Following this seminal work, a number of studies have proposed CTC schemes between Wi-Fi, BLE, and/or IEEE 802.15.4 devices using packet-level information such as the duration [36, 37], interval [6, 7] and transmission power [8, 38] to encode data. Later works exploit the concept of PHY emulation to significantly increase the data rate of CTC [9, 11, 39] or investigate the feasibility of CTC among devices beyond the 2.4 GHz band [40].

However, existing works typically focus on showcasing the viability of CTC (possibly on off-the-shelf devices), or on achieving a high throughput or long range [11,13], mostly neglecting the integration of CTC alongside the functionality of a device. This results in the assumption that all devices know about each other's existence beforehand and that a device can carry out CTC at anytime. Only Hofmann et al. [14] have argued the need to relax these assumptions, but without proposing a tangible solution. SERVOUS fills this gap by proposing – to the best of our knowledge – the first generic cross-technology device discovery and rendezvous protocol, tailoring it to low-power wireless IoT devices.

Device discovery and rendezvous. Neighbour discovery and rendezvous are tightly-coupled problems that have been long investigated by the low-power wireless community [41, 42], among others, in the context of synchronous [16,43] and asynchronous MAC schemes [17, 18]. Prior work in asynchronous neighbour discovery made use of quorum techniques [19, 20] or stochastic approaches (e.g., the use of Birthday protocols [44] to let nodes choose whether to transmit, listen, or sleep with different probabilities). Several protocols, however, require global coordination of the employed duty cycle. Disco is a notable exception in this regard [21]: by scheduling radio wake-up times at multiples of prime numbers, it ensures deterministic pairwise discovery and rendezvous. Similar approaches are followed by U-Connect [22], Searchlight [23], and BlindDate [24].

However, the entirety of existing literature has considered device discovery and/or rendezvous in the context of homogeneous networks employing the same technology. SERVOUS, instead, is the first work studying how to perform discovery and ensure an efficient rendezvous among heterogeneous devices with incompatible PHY – with the additional constraint that a portion of their time cannot be reused in order to not affect ongoing communications. In the context of CTC, only NewBee [45] has touched the topic of neighbour discovery, but with a completely different goal. NewBee, indeed, makes use of PHY emulation to let a Wi-Fi station assist ZigBee devices in finding their neighbours.

10 Conclusions

This paper has presented SERVOUS, a protocol allowing a device to autonomously discover and communicate with surrounding nodes operating on another PHY, while still operating at low duty cycle and without affecting the normal communications of a device. This protocol can be used as basis to enable cooperation between co-located devices besides their normal operations, for example to exchange and coordinate their channel usage, thereby maximizing coexistence. SERVOUS can also be used by co-located devices with incompatible PHY to cooperate and infer the occurrence of specific events, thereby increasing their context awareness.

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/).

11 References

[1] Z. Yu *et al.*, "Crocs: Cross-Technology Clock Synchronization for WiFi and ZigBee," in *Proc. of the* 15th *EWSN Conf.*, 2018.

[2] Z. Yin *et al.*, "Explicit Channel Coordination via Cross-technology Communication," in *Proc. of the* 16th *ACM MobiSys Conf.*, 2018.

[3] I. Rüb *et al.*, "Ad Hoc 802.11-802.15.4 Crosstalk-Based Communication in Practice," in *Proc. of the* 3rd *ACM MadCom Worksh.*, 2018.

[4] Y. Chen *et al.*, "Survey of Cross-Technology Communicationfor IoT Heterogeneous Devices," *IET Communications*, vol. 13, 2019.

[5] K. Chebrolu *et al.*, "Esense: Communication through Energy Sensing," in *Proc. of the* 15th *ACM MobiCom Conf.*, 2009.

[6] X. Zhang *et al.*, "Gap Sense: Lightweight Coordination of Heterogeneous Wireless Devices," in *Proc. of the* 32nd *INFOCOM Conf.*, 2013.

[7] S. M. Kim *et al.*, "FreeBee: Cross-Technology Communication via Free Side-Channel," in *Proc. of the* 21st *ACM MobiCom Conf.*, 2015.

[8] Z. Chi *et al.*, "B2W2: N-way Concurrent Communication for IoT Devices," in *Proc. of the* 14th *ACM SenSys Conf.*, 2015.

[9] Z. Li *et al.*, "WEBee: Physical-Layer Cross-Technology Communication via Emulation," in *Proc. of the* 23rd *ACM MobiCom Conf.*, 2017.

[10] W. Jiang *et al.*, "Achieving Receiver-Side Cross-Technology Communication with Cross-Decoding," in *Proc. of the MobiCom Conf.*, 2018.

[11] ——, "BlueBee: a 10,000x Faster Cross-Technology Communication via PHY Emulation," in *Proc. of the* 15th *ACM SenSys Conf.*, 2017.

[12] S. Wang *et al.*, "Networking Support For Physical-Layer Cross-Technology Communication," in *Proc. of the* 26th *ICNP Conf.*, 2018.

[13] Z. Li *et al.*, "LongBee: Enabling Long-Range Cross-Technology Communication," in *Proc. of the* 37th *IEEE INFOCOM Conf.*, 2018.

[14] R. Hofmann *et al.*, "X-Burst: Enabling Multi-Platform Cross-Technology Communication between Constrained IoT Devices," in *Proc. of the* 16th *IEEE SECON Conf.*, 2019.

[15] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Swedish Institute of Computer Science, Tech. Rep., 2011.

[16] W. Ye *et al.*, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. of the* 21th *IEEE INFOCOM Conf.*, 2002.

[17] J. Polastre *et al.*, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proc. of the* 2nd *ACM SenSys Conf.*, 2004.

[18] M. Buettner *et al.*, "X-MAC: A Short Preamble MAC Protocol for Duty-cycled WSNs," in *Proc. of the* 4th *SenSys Conf.*, 2006.

[19] Y.-C. Tseng *et al.*, "Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks," in *Proc. of the INFOCOM Conf.*, 2002.

[20] S. Lai *et al.*, "Heterogenous Quorum-Based Wake-Up Scheduling in Wireless Sensor Networks," *IEEE Trans. on Comp.*, vol. 59, 2010.

[21] P. Dutta *et al.*, "Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications," in *Proc. of SenSys'08*.

[22] A. Kandhalu *et al.*, "U-connect: A Low-latency Energy-efficient Asynchronous Neighbor Discovery Protocol," in *Proc. of IPSN'10*.

[23] M. Bakht *et al.*, "Searchlight: Won't You Be My Neighbor?" in *Proc. of the* 18th *ACM MobiCom Conf.*, 2012.

[24] K. Wang *et al.*, "BlindDate: A Neighbor Discovery Protocol," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, 2015.

[25] I. Niven *et al.*, *An Introduct. to the Theory of Numbers*. Wiley, 1991.

[26] R. Musaloiu-E. *et al.*, "Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks," in *Proc. of the* 7th *IPSN Conf.*, 2008.

[27] D. Grubmair *et al.*, "Accurate Cross-Technology Clock Synchronization Among Off-The-Shelf Wireless Devices," in *Proc. of the* 17th *EWSN Conf., poster session*, 2020.

[28] T. Todd *et al.*, "Low Power Rendezvous in Embedded Wireless Networks," in *Proc. of the* 1st *MobiHoc Workshop*, 2000.

[29] 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* 17th *EWSN Conf., demo session*, 2020.

[30] SIG Bluetooth, "Specification of the Bluetooth System v5.0," 2016.

[31] M. Spörk *et al.*, "BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices," in *Proc. of SenSys'17*.

[32] T. Puech *et al.*, "A Fully Automated Periodicity Detection in Time Series," in *Proc. of the* 4th *AALTD Workshop*, 2019.

[33] M. G. Elfeky *et al.*, "Using Convolution to Mine Obscure Periodic Patterns in One Pass," in *Proc. of the* 9th *EDBT Conf.*, 2004.

[34] B. M. Elahi *et al.*, "Sensor Ranking: A Primitive for Efficient Content-Based Sensor Search," in *Proc. of the* 8th *IPSN Conf.*, 2009.

[35] T. Schmid *et al.*, "XCXO: An Ultra-low Cost Ultra-high Accuracy Clock System for Wireless Sensor Networks in Harsh Remote Outdoor Environments," in *Proc. of the* 45th *DAC Conf.*, 2008.

[36] Y. Zhang *et al.*, "HoWiES: A Holistic Approach to ZigBee Assisted WiFi Energy Savings in Mobile Devices," in *Proc. of INFOCOM'13*.

[37] S. Yin *et al.*, "Interconnecting WiFi Devices with IEEE 802.15.4 Devices without Using a Gateway," in *Proc. of the DCOSS Conf.*, 2015.

[38] X. Guo *et al.*, "Wizig: Cross-technology Energy Communication over a Noisy Channel," in *Proc. of the 36th INFOCOM Conf.*, 2017.

[39] Y. Chen, Z. Li, and T. He, "TwinBee: Reliable Physical-Layer CTC with Symbol-Level Coding," in *Proc. of the INFOCOM Conf.*, 2018.

[40] P. Gawowicz *et al.*, "Enabling Cross-technology Communication between LTE Unlicensed and WiFi," in *Proc. of INFOCOM*, 2018.

[41] Y. Qiu *et al.*, "Talk More Listen Less: Energy-Efficient Neighbor discovery in WSNs," in *Proc. of the* 35th *IEEE INFOCOM Conf.*, 2016.

[42] P. H. Kindt *et al.*, "Griassdi: Mutually Assisted Slotless Neighbor Discovery," in *Proc. of the* 16th *ACM/IEEE IPSN Conf.*, 2017.

[43] T. Liu *et al.*, "Implementing Software on Resource-constrained Mobile Sensors," in *Proc. of the MobiSys Conf.*, 2004.

[44] M. J. McGlynn *et al.*, "Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks," in *Proc. of the* 2nd *ACM MobiHoc Conf.*, 2001.

[45] D. Gao *et al.*, "Neighbor Discovery based on Cross-Technology Communication for Mobile Applications," *IEEE Trans. Vehic. Tech.*, 2020.