Toro Betancur, Veronica; Bayhan, Suzan; Gawłowicz, Piotr; Di Francesco, Mario

CTC-CEM: Low-Latency Cross-Technology Channel Establishment with Multiple Nodes

Published in:
Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020

DOI:
10.1109/WoWMoM49955.2020.00032

Published: 01/08/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Toro Betancur, V., Bayhan, S., Gawłowicz, P., & Di Francesco, M. (2020). CTC-CEM: Low-Latency Cross-Technology Channel Establishment with Multiple Nodes. In *Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020* (pp. 117-126). Article 9217736 IEEE. https://doi.org/10.1109/WoWMoM49955.2020.00032

CTC-CEM: Low-Latency Cross-Technology Channel Establishment with Multiple Nodes

Verónica Toro-Betancur*, Suzan Bayhan†, Piotr Gawłowicz‡, and Mario Di Francesco*

veronica.torobetancur@aalto.fi, s.bayhan@utwente.nl, gawlowicz@tu-berlin.de, mario.di.francesco@aalto.fi

*Department of Computer Science, Aalto University
†Faculty of EEMCS, University of Twente
‡Telecommunication Networks Group (TKN), Technische Universität Berlin

***Abstract*—Cross-Technology Communication (CTC) allows direct message exchange between devices with different (i.e., incompatible) wireless communication standards. CTC is particularly suitable to allow for coordination between heterogeneous devices sharing the same spectrum, as in the Internet of Things. Existing research on CTC has focused on enabling communications for diverse technologies with the goal of achieving a high throughput. However, it did not address how to establish a link suitable for CTC, which is necessary for successful data exchange. This article specifically addresses such a problem by introducing CTC-CEM (CTC Channel Establishment with Multiple nodes), a scheme to establish a CTC channel involving the use of multiple nodes in a network. CTC-CEM employs duty-cycling and leverages network density to reduce energy consumption, while keeping a low discovery latency. In particular, CTC-CEM defines different discovery protocols to reliably detect co-located networks. Moreover, it addresses the selection of multiple CTC nodes as a set cover problem, and includes an optimization technique based on dynamic programming to balance the energy consumption in the whole network. Extensive simulations show that CTC-CEM effectively distributes the energy consumption in the network, increasing fairness by 97% after optimization. Furthermore, the latency in establishing a channel with CTC-CEM is two orders of magnitude lower than that for device discovery in duty-cycled networks.**

***Index Terms*—Cross-technology communication, channel establishment, network discovery, energy efficiency**

I. Introduction

Cross-Technology Communication (CTC) refers to the direct exchange of messages between devices with different (i.e., incompatible) wireless communication standards. For instance, CTC enables a device with a WiFi interface to send (receive) messages to (from) a device with a Bluetooth transceiver, without the need for additional hardware [1]. CTC is particularly suitable to allow for coordination between heterogeneous devices sharing the same spectrum, particularly, in the unlicensed ISM bands. This is especially important in the Internet of Things, wherein a large number of devices is densely deployed in a certain geographical area. In such scenarios, devices could leverage CTC to effectively mitigate cross-technology interference and increase performance as well as reliability.

CTC is achieved by operating at either the physical layer or the packet level. Solutions of the first type emulate signals of another technology [2, 3]. Packet-level schemes, instead, leverage capabilities that are common to different technologies (such as frequency, phase and amplitude sensing) to encode messages [1, 4–7]. In any case, CTC requires establishing a logical *link* – usually called *channel* – in the first place. However, existing research on CTC has rather focused on enabling communications for diverse technologies (e.g., WiFi-ZigBee and WiFi-Bluetooth) with the goal of achieving a high throughput (Section II). In doing so, most solutions rely on the fact that a CTC channel is already available, without providing means to establish such. Moreover, they only employ a single designated node in each network (e.g., an access point or coordinator) to realize CTC, resulting in limited coverage and possible traffic bottlenecks.

Indeed, establishing a CTC channel entails two key challenges. First, a given network should quickly discover the presence of others using a different technology. This also requires designing a scheme that is robust against message losses and has a limited overhead, especially for dynamic environments demanding continuous network discovery. Second, nodes incur a higher energy expenditure due to CTC, especially coordinators that need to handle traffic belonging to the whole network. Leveraging multiple devices balances the energy expenditure across nodes in the network; moreover, it increases efficiency as devices can communicate directly instead of through coordinators. However, extending CTC to more than one node per network requires careful coordination for selecting devices and scheduling their transmissions.

We specifically address these challenges by introducing CTC-CEM (CTC Channel Establishment with Multiple nodes), a scheme to establish a CTC channel involving the use of multiple nodes in a network (Section III). CTC-CEM is transparent, namely, it can be applied to a variety of CTC protocols, including C-Morse [4] and DopplerFi [5]. Moreover, CTC-CEM employs duty-cycling and leverages network density to reduce energy consumption, while keeping a low discovery latency. In particular, CTC-CEM defines discovery protocols to reliably discover networks using different communication technologies under the presence of a duty cycle (Section IV). Moreover, it addresses the selection of multiple CTC nodes as a set cover problem, and includes an optimization technique based on dynamic programming to balance the energy consumption in the whole network (Section V). Extensive simulations show that CTC-CEM effectively distributes the energy consumption in the network,

978-1-7281-7374-0/20/$31.00

increasing fairness by 97% after optimization. Furthermore, the latency in establishing a channel with CTC-CEM is two orders of magnitude lower than that for device discovery in duty-cycled networks (Section VI).

II. Related work

Prior work in the literature has introduced several approaches for CTC at both the physical layer and the packet level. Among those of the first class, WEBee [2] proposes parallel CTC transmissions on different frequency bands by emulating ZigBee signals in WiFi commodity devices. BlueBee [3] addresses CTC between Bluetooth and ZigBee. Packet-level CTC has also been extensively addressed. C-Morse [4] encodes messages through the packet length as a Morse code. Specifically, a small packet length represents a dot, while a larger packet or two consecutive packets represent a dash. DopplerFi [5] introduces artificial Doppler shifts to encode CTC messages in frequency so as to enable direct communication between BLE and WiFi devices. ZigFi [7] presents a ZigBee to WiFi communication scheme that uses channel state information to decode CTC messages. The same work also proposes an initial protocol to establish communication parameters, such as transmission power and packet length; however, the network discovery problem is not addressed. In contrast to the prior work in CTC, we address network discovery, the establishment of the CTC channel and propose the use of multiple nodes to carry CTC messages.

Device discovery and wireless channel establishment have also been addressed, however, in other scenarios – primarily in the literature about wireless sensors and opportunistic networks [8, 9]. In this context, nodes perform continuous device discovery while using a duty-cycle to save energy. There are two main approaches for opportunistic discovery: probabilistic and deterministic. The family of birthday protocols proposed by McGlynn et al. [10] belong to the first group: time is divided into periods and each node independently and randomly chooses slots in the period to transmit or listen. Karowski et al. [11] use linear programming optimization for asynchronous multi-channel neighbor discovery.

Deterministic protocols provide a latency bound: nodes are able to discover each other in a finite amount of time. Zheng et al. [12] provide lower bounds for the discovery problem by proposing an asynchronous wakeup scheduling based on block design. Later, Meng et al. [13] introduce Diff-codes to prove that such bounds can be further lowered by considering non-aligned slots. Different from traditional approaches, Chen et al. [14] do not assume slotted time. Wei et al. [15] propose two types of time slots with different active lengths to achieve better energy savings. However, these approaches target non-CTC applications and leverage device-to-device communication. In contrast, CTC-CEM uses all nodes to achieve a lower discovery latency and to better balance energy consumption of nodes in heterogeneous networks.

III. System Model

We consider two wireless networks with overlapping coverage areas, namely, some nodes in each network sense transmissions from the nodes in the other network, generally called *foreign* network in the literature. Each network has a *controller* managing the use of network resources. For illustration purposes, the rest of the discussion assumes a WiFi network co-located with a ZigBee network; accordingly, the controllers are represented by a WiFi access point (AP) and a ZigBee coordinator (ZC). Moreover, we address the general case where the controllers cannot communicate with each other directly; instead, other nodes in the network must be used for CTC. The two networks operate at the same or partially overlapping spectrum, resulting in co-channel interference. Networks establish a CTC channel, for instance, to mitigate such interference; therefore, they perform network discovery upon detecting frequent packet drops. We assume that both networks run the same CTC scheme and use special *CTC control frames*: *synchronization* (SYN) messages and ACK messages (i.e., frames without payload) acknowledging them. Moreover, each network controller divides nodes into two types: those transmitting network discovery messages; and those listening to network discovery messages from the foreign network. We refer to nodes of these two types as *SYN transmitters* and *SYN receivers*, respectively, and to all of them as *SYN nodes*.

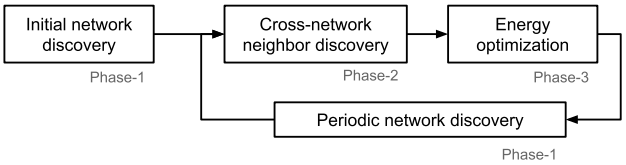


Fig. 1: The different phases in CTC-CEM.

Before establishing the CTC channel, a network becomes aware of the other one through a Cross-Technology Interference (CTI) detection mechanism, for instance, one of those in [16–19]. Then, the channel establishment takes place through CTC-CEM in three phases: (i) network discovery (Section IV), (ii) cross-network neighbor discovery (Section V-A), and (iii) energy optimization (Section V-C). These phases are illustrated in Fig. 1.

IV. Foreign network discovery

As mentioned in Section III, we assume that both networks run the same CTC scheme. However, the networks do not have any knowledge on the characteristics of the other one, for instance, its communication technology. Therefore, they start a *foreign* network discovery process. To this end, the controller defines a duty-cycle consisting of two types of time slots: *active slots* during which nodes perform network discovery and *sleeping slots* during which nodes either save energy or operate as usual for their in-technology communication. There are two types of active slots: a Tx slot for SYN transmitters and a Rx slot for SYN receivers.

Network discovery is complete after the two networks successfully exchange network discovery messages. For this to happen, the following two conditions must hold: (i) an active Tx slot of a network must overlap with an active Rx slot of the other network long enough for the network discovery

message to be decoded, and (ii) at least one SYN receiver must be in the transmission range of at least one SYN transmitter. However, networks likely start the network discovery process at different times as the impact of interference on each network is generally asymmetric. Moreover, the location or availability of nodes might change after the CTC channel establishment. Therefore, CTC-CEM must run periodically to address these issues and ensure efficient operations. For this reason, we propose two different network discovery schemes: an *initial network discovery* and a *periodic network discovery*. We introduce these next.

A. Initial network discovery

Since networks may start the initial network discovery phase at completely different times, it is necessary to use a neighbor discovery scheme with a low duty-cycle to save energy while still guaranteeing network discovery. Particularly, we leverage existing deterministic neighbor discovery schemes (such as Diff-codes [13] or the solution in [12]) that guarantee overlapping active slots in a finite amount of time. In the following, we describe how to apply such schemes in the considered CTC scenario.

Since the controllers of each network do not know which nodes are in the cross-network transmission range, they select different random sets of nodes as SYN transmitters and SYN receivers at the beginning of each active slot. This guarantees the detection of discovery messages. Such messages include a technology-specific code, referred to as *SYN code*, to identify the technology type of the network. Such codes are defined by the CTC scheme. For instance, CMorse [4] defines a SYN code as a certain sequence of dots and dashes. Instead, DopplerFi [5] employs different values of artificial Doppler shifts. A network determines the foreign technology by decoding the received SYN code.

As Fig. 2a shows, an active Tx slot (T) is composed by two portions (i.e., Tx sub-slots): the first one, with a duration of T_S, for sending the SYN code; the second one, with a duration of T_A, for listening to ACKs and the SYN code from the foreign network. Clearly, $T = T_S + T_A$ and generally $T_S < T_A$. The selected set of SYN receivers are constantly listening for foreign SYN codes during an active Rx slot. The exact duration of Tx and Rx slots is defined by the applied CTC scheme, as SYN codes are represented accordingly. During the initial discovery phase, both networks run Tx and Rx slots in parallel with different sets of nodes. The process continues until a SYN code of the foreign network is detected. The energy consumption is distributed among the network by selecting different sets of SYN transmitters and receivers for each slot.

The Rx slot is longer than the Tx sub-slot, so that the SYN receivers of the foreign network can detect the SYN message. Let us denote the length of the Rx slot by kT, where $k > 1$ defines the lower bound for neighbor discovery. We set $k = 2$ to keep the neighbor discovery latency bounded by two Tx sub-slots. As a result, an active Tx slot (composed by two Tx sub-slots) and an active Rx slot have the same length. Note that slots must have the same length to ensure overlaps under deterministic neighbor discovery approaches, such as those based on block designs [12].

The two networks perform a three-way handshake to establish the CTC channel. Let us consider a sample scenario of ZigBee and WiFi networks. For simplicity, we consider only the Tx slots of the ZigBee network and the Rx slots of the WiFi network. The following details the three-way handshake in the considered scenario.

1) The ZigBee SYN code is detected[1] by some WiFi SYN receivers. These WiFi nodes immediately inform the WiFi AP about the received code.
2) The WiFi AP synchronizes all nodes to send an ACK followed by the SYN code corresponding to the network technology type. During this second step, the ZigBee nodes that sent the detected code are in listening mode waiting for the WiFi ACK and SYN code. After these ZigBee nodes detect the WiFi SYN code, they inform the ZC about it.
3) The ZC then synchronizes all nodes to send an ACK to the WiFi network.

Let us now analyze the discovery latency in this scenario. First of all, we note that network discovery can be completed only if an active Tx and Rx slots overlap and at least one SYN receiver is in transmission range of at least one SYN transmitter of the foreign network. However, it is possible that two active slots overlap for insufficient time, thus the SYN code is not successfully received. Recall that an active Tx slot is composed by two Tx sub-slots. Consider also, for instance, the ZigBee Tx slots and the WiFi Rx slots shown in Fig. 2a. Since networks are unsynchronized in their active slots, four different *time shifts* between these Rx and Tx slots can occur in this considered setting. These cases are as follows.

a) The WiFi Rx slot starts before the first ZigBee Tx sub-slot and the WiFi active Rx slot ends before the first ZigBee SYN slot. As a result, the WiFi Rx slot cannot detect the ZigBee SYN code.
b) The Rx slot starts before the first Tx sub-slot. However, WiFi nodes are still able to detect the first SYN code.
c) An Rx slot starts after the first SYN code, which allows to detect the second SYN code entirely.
d) An Rx slot starts after the second Tx sub-slot, which makes the nodes miss all SYN codes.

Since active and sleeping slots have the same duration, the time shift is constant. As a result, networks will never discover each other in Case a) and Case d). Therefore, we extend the active Rx slots as shown in Fig. 2b: if the i^{th} active slot is allocated to Rx (Rx_i), then slots $i-1$ and $i+1$ must also be active. As a result, networks always meet at a partially overlapping active slot if the SYN receivers are in coverage area of the SYN transmitters. Consequently, the discovery latency is entirely determined by the employed neighbor discovery approach.

[1]This happens as soon as a ZigBee Tx slot and a WiFi Rx slot overlap.

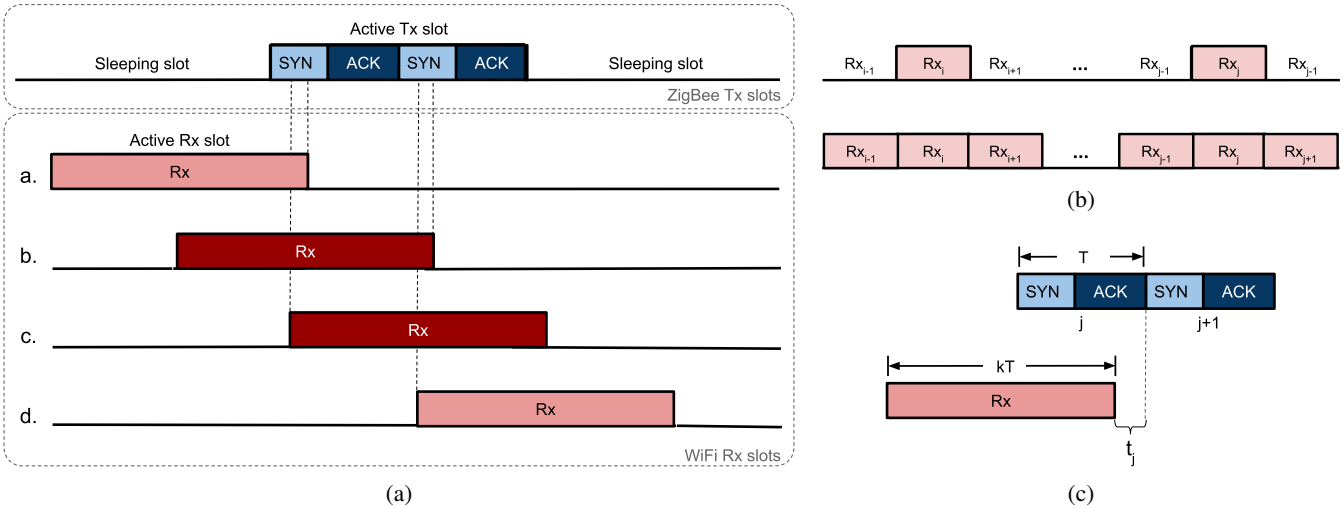


Fig. 2: (a) Sample time shifts between Tx slots of the ZigBee network and Rx slots of the WiFi network, (b) extension of active Rx slots and (c) general discovery latency.

B. Periodic network discovery

CTC-CEM must run periodically to address network dynamics due to either mobility or node failures. However, running a neighbor discovery scheme as described in Section IV-A all the time is not efficient. For this reason, we establish a periodic schedule at which point the two networks start running a periodic network discovery process. Specifically, we introduce CTC-PND (CTC Periodic Network Discovery), an asynchronous discovery approach employing all nodes in the networks and leveraging different sets of nodes in each active slot, as in the *initial* network discovery approach presented above. However, nodes send and receive SYN codes without going to a sleep state until a certain condition is met for the *periodic* network discovery. Since the two networks have already agreed on a schedule, a low duty-cycle is no longer needed.

Let us calculate the lower discovery bound for CTC-PND. Consider the scheme in Fig. 2c, wherein Tx slots have a duration of $T = T_S + T_A$ with $T_S < T_A$ and the Rx slot lasts k times longer than the Tx slot (kT). Let t_j be the time shift measured with respect to the end of the j^{th} Tx slot: $t_j = R_j - T_j$ where R_j and T_j are the ends of the j^{th} Rx and Tx slots, respectively, as shown in Fig. 2c. In this scenario, the condition for the j^{th} SYN code to be detected depends on whether t_j is negative or positive and is given by:

$$-T_A \leq t_j < 0 \quad \text{or} \quad 0 \leq t_j \leq T(k-1), \tag{1}$$

respectively. Accordingly, the discovery latency is defined by:

$$DL = \begin{cases} \left\lceil \frac{||t_j| - T_A|}{T(k-1)} \right\rceil & \text{if} \quad t_j < 0 \\ \left\lceil \frac{t_j}{T(k-1)} \right\rceil & \text{if} \quad t_j \geq 0 \end{cases} \tag{2}$$

This equation represents the difference between the time shift and the lower bound in the detection condition ($||t_1| - T_A|$ or $|t_1 - 0|$), as given in Eq. (1), divided by the amount that this difference is shortened in each slot ($T(k-1)$). It is clear from Eq. (2) that for small values of k (i.e., $1 < k < 2$) the discovery latency increases, while for $k \geq 2$ the lower bound is always equal to two slots. Clearly, $k > 1$, otherwise the discovery condition is never met unless $k = 1$ and the slots are completely aligned.

CTC-PND follows the same scheme as in Fig. 2c, where $k = 2$. Hence, the condition for the j^{th} SYN code to be detected is $-T_A \leq t_j < 0$, or $0 \leq t_j \leq T$, and, according to Eq. (2), the maximum discovery latency is two slots.

Nevertheless, this lower bound applies to a scenario with a pair of SYN transmitter and receiver, within their transmission ranges, continuously running Tx and Rx slots. However, networks do not know yet which nodes are in the cross-network transmission range. Moreover, the discussed lower bound is achieved if all nodes are scheduled with Tx and Rx slots, at the cost of a high energy consumption. Hence, the networks continuously listen and transmit while distributing the energy between all the nodes by allowing each controller node to randomly choose the nodes for receiving and transmitting SYN codes in such a way that each Tx and Rx slot is run by a different set of nodes. We show in Section VI-B that such an approach is more efficient than using a traditional deterministic neighbor discovery protocol in a periodic network discovery process.

C. Implementation-related considerations

We use multiple nodes to transmit a CTC message in the same active time-slot to increase the probability of successful reception of the CTC message by at least one node in the foreign-technology network. Depending on the

implementation of the underlying CTC scheme, the nodes may transmit CTC messages in different ways: sequentially in the case of emulation-based CTC schemes, where the entire CTC message is emulated and contained in a single frame; or exactly at the same time in the case of Packet-level CTC schemes, where multiple frames are used to encode CTC message using parameters (such as frame duration, transmission power and so on). In either case, time synchronization among nodes generating CTC messages is required. Note that the most widely used communication technologies maintain a time synchronization between the nodes in the network. For instance, the time synchronization function (TSF) in WiFi keeps the timers for all stations in the same basic service set synchronized, i.e., all WiFi stations set their local TSF time to that announced by the AP. Existing works in the literature, such as [20, 21], have shown that it is possible to synchronize actions performed by WiFi nodes based on their TSF timers. Furthermore, forcing multiple nodes to transmit at exactly the same time requires deactivating the physical and virtual carrier-sensing mechanisms that make the nodes backoff in case some activity on the channel is detected. This is actually possible in at least some off-the-shelf WiFi chipsets. Therefore, it is feasible to implement the proposed CTC channel establishment scheme on top of existing wireless standards, particularly, WiFi.

V. Neighbor discovery and energy optimization

At this stage, each network is aware of the existence of the other network and its technology type. However, the controllers do not know how many foreign nodes are in the coverage range and which local nodes are able to reach foreign nodes. This information is crucial to select CTC transmitters and receivers for ensuring reception of CTC messages. Moreover, such a selection should reduce the energy consumption in the whole network. These aspects are detailed next.

A. Cross-network neighbor discovery

The second phase in CTC-CEM (recall Figure 1) is cross-network neighbor discovery. The controller initiates such a phase, then each node broadcasts its identity once and waits for ACK messages from the foreign nodes. In this case, all nodes send a CTC control frame sequentially. Consider again our reference ZigBee-WiFi scenario. Since the ZigBee nodes sent the last ACK during the foreign network discovery phase, now the WiFi network needs to identify its nodes. While one WiFi node sends a CTC control frame, the remaining nodes sense to detect ACK messages from the ZigBee network. After receiving an ACK for its message, the node knows that its transmission was successfully received by the ZigBee network. Therefore, this node is a candidate to carry CTC transmissions. The AP is in charge of creating a TX-CTC table with all the local nodes that are capable of transmitting CTC messages.

On the other hand, the ZigBee network must identify the nodes that are able to receive the WiFi transmissions. For this purpose, the ZC needs to create an RX-CTC table with all the WiFi nodes that each ZigBee node senses. To this end, every time a ZigBee node reports a successful CTC transmission by WiFi nodes, the ZC assigns an ID to the WiFi node and adds it to the local node entry in the RX-CTC table. Note that the ID that the ZC assigns to the WiFi nodes is freely chosen. Hence, it is not necessary to include any kind of ID for the nodes in the CTC control frames. Moreover, since nodes send control frames only once during this phase, they cannot be counted more than once. Table I shows a sample RX-CTC table. In this example, ZigBee node Z_2 can only sense transmissions coming from WiFi node W_4, while node Z_4 is not reachable by any WiFi node.

TABLE I: Sample RX-CTC table.

RX-CTC table
$Z_1 = \{W_1, W_3, W_4, W_5\}$
$Z_2 = \{W_4\}$
$Z_3 = \{W_1, W_2\}$
$Z_4 = \{\ \}$

Both TX-CTC and RX-CTC tables are created locally and it is not necessary to share them with the other network. While nodes in TX-CTC are immediately considered as CTC transmitters, the RX-CTC table cannot be directly used to determine CTC receivers since it is not possible for a network to know which nodes will carry a CTC message in the foreign network. Instead, the controller needs to calculate sets of nodes that cover all foreign nodes to ensure that a CTC message is received by at least one local node. The example in Table I illustrates that no node covers all foreign nodes. However, the set $\{Z_1, Z_3\}$ forms a *set cover*: if these nodes sense simultaneously, the controller guarantees that all CTC transmissions from the WiFi network are received.

The set cover problem is a well-known NP-complete problem [22–24]. CTC-CEM implements a typical set cover algorithm to find the set covers of CTC receivers. However, we propose a new heuristic to achieve fair energy consumption among the involved nodes. Next, we present our approach to CTC receiver selection.

B. Set cover-based CTC receiver selection

Now, each controller knows which of its local nodes sense which foreign nodes. With this information, a controller needs to determine the set of nodes that are able to act as CTC receivers at a particular time. We abstract our problem as a weighted set cover problem, since using always the same nodes as CTC receivers leads to an unfair energy consumption at these CTC receivers. In contrast to static node weights [25], we propose using dynamic weights (DWs) that are increased every time a node is included in a set cover. This allows to distribute all nodes in the set covers instead of including always the same. The fair distribution of nodes in the set covers is essential to ensure that no node is exhausted by high energy consumption. We discuss next the implementation of DWs.

We propose CTC-set covers in Alg. 1 to generate set covers of nodes such that all possible transmissions from the foreign network are sensed. The input parameters of this algorithm

Algorithm 1: CTC-set covers

```
Init : E ← all foreign nodes, N ← all local nodes
       U ← {e|e ∈ E} iff ∃n ∈
       N s.t. e is covered by n  n ← max_n |U ∩ n|
       s ← ∅, C ← ∅, w(n) ← 0        ∀n ∈ N
1  Procedure SetCovering(U, N, n, s, C, w)
2      s.add(n), N.remove(n);
3      foreach elements covered by n do
4          U.remove(elements)
5      if U is empty
6          s ← RemoveRedundant(s);
7          if s is not in C then C.append(s);
8          foreach m in s do  w(m) ← w(m) + 1;
9          s ← ∅;
10         return C
11     else
12         sort N by w(n_i)/|U ∩ n_i| ∀n_i ∈ N;
13         foreach n' in N do
14             SetCovering(U, N, n', s, C, w);
15         s ← ∅;
16     return C
```

are a list of nodes that have not yet been included in a set cover ($\mathcal{N}$), the universe ($\mathcal{U}$) representing all foreign nodes that are sensed by at least one local node, a node n to be included in the current set cover s, the current list of all set covers ($\mathcal{C}$), and the current weights w of all nodes. We first set the weights for all nodes to zero. Note that n is the node achieving the maximum intersection with the universe $\mathcal{U}$.

This recursive algorithm starts by adding node n to the set s and removing it from the list of available nodes (line 2). In line 4, all elements covered by n are removed from $\mathcal{U}$. Then, the base case is checked: if $\mathcal{U}$ is empty, i.e., all foreign nodes are covered. Next, RemoveRedundant removes all possible redundant nodes in the generated set cover using the algorithm in [22]. RemoveRedundant iterates over all nodes in the set. In case all elements covered by a node are also covered by the rest of the nodes, such node is considered redundant and, thus, removed from the set cover. It is also necessary to verify if the set cover s is not already in the set of set covers. This verification occurs in line 7 after which s is appended to $\mathcal{C}$. The next step is to increase the weight of the nodes included in the set cover (line 8). Moreover, s is set to an empty set and $\mathcal{C}$ is returned. However, if there are still nodes in $\mathcal{U}$ that need to be covered, the remaining nodes ($\mathcal{N}$) are sorted according to:

$$\mathcal{H} = \frac{w(n_i)}{|\mathcal{U} \cap n_i|}. \tag{3}$$

This heuristic uses the nodes' current weight and the length of the intersection between the elements covered by each node and the universe $\mathcal{U}$. As a result, the nodes with the smallest weight and largest intersection length are chosen first. The last step in CTC-set covers is to recurse over SetCovering with node (n') in line 14. After finding the set covers, the controller node can set the nodes that belong to any set cover in listening mode to listen to foreign CTC transmissions. Note that the procedure in Alg. 1 is a recursive approach to the exact set cover enumeration problem [22] with a heuristic that achieves fair distribution of nodes in the set covers. The correctness of CTC-set covers is proved similarly as in typical algorithms for finding set covers; thus, we omit it here due to limited space.

C. Energy optimization

After the set covers have been generated, the controller needs to determine which of them to use, as using all set covers is not always the most efficient solution in terms of fair energy balance of the nodes. Therefore, the controller aims at using given nodes approximately the same number of times.

Let $\mathcal{C}$ be the set of set covers (s) discovered in the previous step, and w_i be the number of times a node n_i appears in a subset. Our goal is to choose some subsets of $\mathcal{C}$ such that $w_i \approx w_k \ \forall i,k \in \mathcal{N}$. Specifically, we define the energy optimization problem as minimizing the sum of the difference of w between every pair of nodes, as shown in Eq. (4). Let us denote the sum of differences by G which represents the balance of workload among the local nodes. We define the objective function of our energy optimization problem as follows:

$$\min\{G\} = \min \left(\sum_{i=1}^{|\mathcal{N}|-1} \sum_{k=i+1}^{|\mathcal{N}|} |w_i - w_k| \right). \tag{4}$$

Let S be a schedule composed by all the set covers generated during the cross-network neighbor discovery phase. That is, $S = \mathcal{C}$. However, the schedule S also defines a policy for using the set covers cyclically. It is possible to find another schedule S' such that $S' \subseteq S$ and $G^{(S')} \leq G^{(S)}$ for all iterations. Particularly, we propose the dynamic programming approach presented in Alg. 2.

Alg. 2 uses a greedy heuristic to always include the set cover that minimizes the sum in Eq. (4). The dynamic programming algorithm creates an improved schedule S' by adding the set covers that minimize G until all nodes in $\mathcal{N}$ are covered. Specifically, the algorithm increases the weights (w) of the nodes every time they are included in the schedule S'. Then, it uses two dictionaries (N and Z) to store the weight difference between every pair of nodes. N contains an entry for each node in $\mathcal{N}$ and a node n_j is included in the entry of another node n_i if $w(n_j) > w(n_i)$. On the other hand, Z also contains an entry for each node in $\mathcal{N}$. In this dictionary, nodes are included in another node's entry only if their weights are equal, i.e., $w(n_j) = w(n_i)$. With this data structure, Eq. (5) can be used every time one needs to calculate the contribution to G of adding a new set cover s to S'. There, G_{k+1} is the sum G after including a new set while G_k is the sum in the previous iteration of the algorithm and n_l is the number of nodes, i.e., $n_l = |\mathcal{N}|$. Here, $|s|$ denotes the number of nodes in the set cover s while m is the sum of number of elements in the entry of node n in the dictionary

Algorithm 2: Find improved schedule

Input : $\mathcal{C}$: List of current set covers; $\mathcal{N}$: List of nodes
Output : S': Improved schedule
Init : $S' \leftarrow \emptyset, G_k \leftarrow 0, n_l \leftarrow |\mathcal{N}|$, $w(n) \leftarrow 0$, $N(n) \leftarrow \emptyset$, $Z(n) \leftarrow \mathcal{N}$ $\quad \forall n \in \mathcal{N}$

1 **while** $\mathcal{N} \neq \emptyset$
2 $G_{k+1}(s) \leftarrow 0 \quad \forall s \in \mathcal{C}$
3 **foreach** $s \in \mathcal{C}$ **do**
4 $m \leftarrow \sum_{i=0}^{|s|} |N(i)|$
5 $G_{k+1}(s) \leftarrow G_k + n_l|s| - 2m - |s|$
6 $s' \leftarrow G_{k+1}$.argmin(), $G_k \leftarrow G_{k+1}(s')$
7 S'.add(s'), $\mathcal{C}$.remove(s')
8 **foreach** $n \in s'$ **do**
9 $\mathcal{N}$.remove(n)
10 **foreach** $k_1 \in Z(n)$ **do**
11 $N(k_1)$.add(n), $Z(k_1)$.remove(n)
12 $Z(n) \leftarrow \emptyset$, $w(n) \leftarrow w(n) + 1$
13 **foreach** $k_2 \in N(n)$ **do**
14 **if** $w(n) == w(k_2)$
15 $Z(n)$.add(k_2) and $Z(k_2)$.add(n)
16 $N(n)$.remove(k_2)
17 **return** S'

N for all $n \in s$, i.e. $m = \sum_{n \in s} |N(n)|$. We express G_{k+1} as follows:

$$G_{k+1} = G_k + n_l|s| - 2m - |s|. \tag{5}$$

Alg. 2 initializes the schedule S' to an empty set, $G_k = 0$ and $n_l = |\mathcal{N}|$; the weights of all nodes to zero, all entries of N as empty sets, all entries of Z as $\mathcal{N}$, because all nodes have weight zero. Then, a *while* loop is used to run the algorithm until $\mathcal{N}$ is empty (line 1). G_{k+1} is initialized as an array that contains the result of G if a set s is included in S' (line 2). G is then evaluated for each set in $\mathcal{C}$ by means of Eq. (5) (lines 3-5). The set that produces the smallest sum is added to the schedule S' and removed from $\mathcal{C}$ (lines 6-7). At this point, N and Z must be updated. All nodes that are included in s' are removed from $\mathcal{N}$ (line 9). Afterwards, before increasing the weight of n, all nodes that has equal weight as n are removed from the related lists in Z and n is added to their lists in N (lines 10-11). The weight of n is increased by one (line 12); if another node (k_2) has the same weight as n, that node is removed from $N(n)$ and k_2 and n are added to the related lists in Z (lines 13–16). Finally, the algorithm returns the improved schedule S' (line 17). Alg. 2 runs in $O(|\mathcal{N}||\mathcal{C}| + |\mathcal{N}|^3)$. If $|\mathcal{C}| \leq |\mathcal{N}|$ then the time complexity becomes $O(|\mathcal{N}|^3)$.

The energy optimization phase should run every time the set covers change. For this reason, the periodic network discovery approach presented in Section IV-B, followed by the other phases, must run regularly as shown in Fig. 1. Periodic operation allows keeping the set covers updated to avoid the use of CTC receivers that cannot sense any foreign node any more.

D. Latency analysis

The latency introduced by phase 1 was already discussed in Section IV. In addition to this latency, phases 2 and 3 also require time to be executed, thus, they contribute to the total latency of CTC-CEM. Let us first analyze the number of messages sent during phase 2. The number of messages clearly depends on the number of nodes in both networks. Assume two networks composed by n_1 and n_2 nodes each. The total number of CTC control frames sent by both networks is $n_1 + n_2$, while the number of ACKs is at most $n_1 + n_2$. The maximum number of ACKs is reached when all nodes are sensed by at least one foreign node. Hence, the total number of messages sent during phase-2 (s) is given by:

$$n_1 + n_2 \leq s \leq 2(n_1 + n_2). \tag{6}$$

Assuming that it takes t time units to send each message, the contribution of this phase to the total latency is st. Additionally, the controllers in each network run Alg. 1 during phase 2. Let us then analyze its time complexity. CTC-set covers finds all set covers with exponential time complexity. However, one can reduce the complexity by setting a threshold (t) that limits the number of generated set covers. This threshold reduces the branching factor of the search tree to t, resulting in a time complexity of $O(|\mathcal{N}|^t)$, where $|\mathcal{N}|$ is the number of local nodes. Note that t is chosen depending on the processing capabilities available and can be as small as two, which leads to a time complexity of $O(|\mathcal{N}|^2)$. This is particularly useful for very large networks with thousands of nodes, since coordinators may not have the required processing capabilities to calculate all set covers.

Finally, no messages are sent in phase 3 but the controller node still needs to run Alg. 2. Let us now derive the time complexity in this case. The use of dictionaries N and Z in Alg. 2 allows to use the recurrence in Eq. (5) to calculate G_{k+1} through G_k. Hence, it is not necessary to use Eq. (4) which runs in $O(|\mathcal{N}|^2)$. As a result, a problem with $O(|\mathcal{N}|^2)$ complexity is solved in $O(|s|)$, which is $O(|\mathcal{N}|)$ in the worst case.

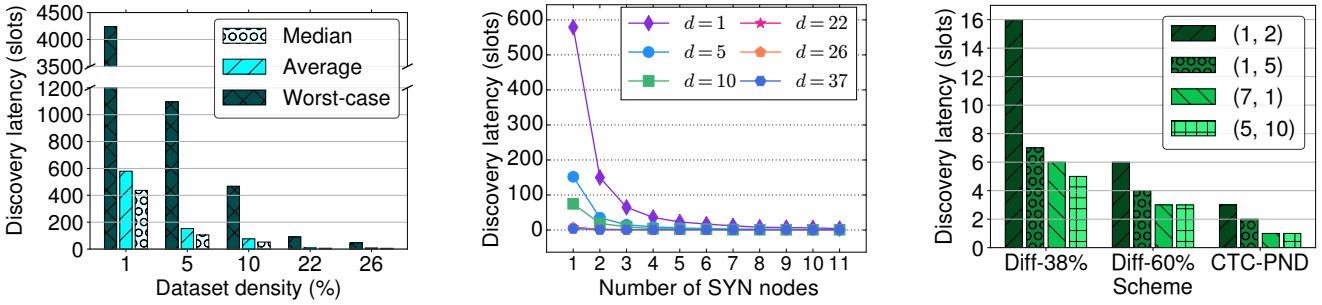
Note that both networks are dedicated to CTC-CEM during phases 1 and 2, and all CTC messages are dummy packets (i.e., packets with no payload). As a result, the network data transmissions are delayed, unless in-network data packets can be scheduled to be sent at times that coincide with the CTC-CEM process. In that case, CTC-CEM runs without delaying network transmissions.

VI. Performance evaluation

This section evaluates the performance of CTC-CEM through extensive simulations. First, we introduce the methodology and setup. Then, we present the results related to both network discovery and balancing energy consumption.

A. Simulation setup and methodology

We use a custom Python simulator and generate two multi-hop (WiFi and ZigBee) networks, wherein the controller



(a) (b) (c)

Fig. 3: Discovery latency under different dataset densities for (a) a single SYN node, (b) a varying number of SYN nodes and (c) in the asymmetric case compared to Diff-codes.

TABLE II: Simulated CTC scenarios.

Deployed area (km x km)	Number of nodes		Dataset density (%)	
	WiFi	ZigBee	WiFi	ZigBee
0.6 x 0.6	20	50	10.75	1.00
0.8 x 0.8	20	50	5.00	99.40
1.9 x 1.9	20	50	1.50	87.40
2.5 x 2.5	20	50	0.50	70.80
3.0 x 3.0	20	40	0.25	55.00
4.0 x 4.0	20	40	0.25	37.87
5.0 x 5.0	20	40	0.00	26.37
5.5 x 5.5	20	80	19.56	22.00

of each network is outside the coverage area of the other network. The simulator deploys the nodes uniformly in the considered area, then determines which foreign nodes can be detected by each local node by using log-normal shadowing and a Rayleigh fading model, with the variance parameter set to one in both cases. Finally, the input for the set cover problem is generated.

The simulation uses the transmission power and receiver sensitivity of representative WiFi and ZigBee devices. For the WiFi devices, we consider the Atheros AR9271 chipset with a transmission power of 27 dBm and a receiver sensitivity of -91 dBm, which are the nominal values for the IEEE 802.11n specification at a rate of 6.5 Mbps. For the ZigBee devices, we consider a Digi XBee S2C module with a Tx power of 3.1 dBm and a sensitivity of -100 dBm. The model also assumes omnidirectional 0 dBi-gain antennas for all nodes. With these parameters, WiFi nodes have a coverage range of approximately 350 m, while ZigBee nodes have a coverage range of 140 m.

We consider different deployment areas and number of nodes for each network by generating different synthetic datasets. Table II shows some of the simulation parameters and the obtained dataset densities for each network. The dataset density term in the table indicates the density of the set cover matrix, namely, the sum of covered elements in the RX-CTC table.

We use the datasets to evaluate CTC-CEM. Specifically, we first investigate how the number of SYN nodes and the dataset density affect the discovery latency. We then study the distribution of energy among the nodes when using dynamic weights (as explained in Section V-B) and improved schedules (as discussed in Section V-C).

B. Network discovery

Clearly, the discovery latency for the initial network discovery phase (Section IV-A) is defined by the used neighbor discovery approach. Instead, we consider different scenarios by varying the parameters in Section VI-A to evaluate the discovery latency of the periodic network discovery phase (Section IV-B). Depending on the locations of the nodes, we obtain set cover matrices with different densities. For each dataset, we evaluate the performance of CTC-PND by varying the number of SYN nodes in each network.

Fig. 3a shows the discovery latency for low dataset densities, i.e., up to 26%. In these scenarios, we set the number of SYN transmitters and receivers to one and record the worst-case, average, and the median from 1,000 iterations for each dataset density. The worst-case discovery latency in the figure differs considerably from the average and median latencies. This is due to the random selection of SYN transmitters and receivers. However, the average and median values show that the networks usually discover each other much sooner than the worst-case scenario.

Fig. 3b illustrates the average discovery latencies for different numbers of SYN nodes. Each line in the figure corresponds to a certain dataset density (d). The figure highlights a significant decrease in the discovery latency when two SYN nodes are used instead of a single SYN node, which confirms the merit of CTC-CEM. In particular, the benefit of using multiple nodes becomes more visible under low densities. We also observe that only 6 SYN nodes are needed to achieve average discovery latencies of 15 time slots or less, for all the considered densities. Moreover, the average discovery latency rapidly reaches a steady-state in a few time slots for densities of at least 5%. Specifically, such a state is reached by using 10 SYN nodes or more. This is due to the fact that the discovery probability increases with the number of SYN nodes, since more nodes are in the cross-network coverage area.

We also analyze the case of asymmetric network discovery in Fig. 3c. In this case, each controller decides on the number of SYN nodes independently, which leads to a scenario where

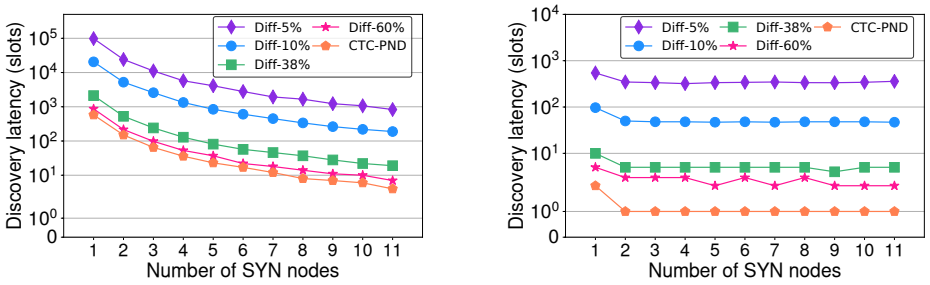


Fig. 4: Average discovery latency vs number of SYN nodes in each network for Diff-codes with different duty-cycles and CTC-PND with dataset densities of (a) 1% and (b) 55%.

each network chooses a different number of SYN nodes. The notation (1, 2) in the figure corresponds to the case where the local network chooses one SYN node and the foreign network chooses two SYN nodes. Similarly, we consider settings of (1, 5), (7, 1), and (5, 10). Additionally, we consider the same scenarios using Diff-codes. To this end, we derive codes by using the algorithms presented in [13] and obtain duty-cycles of 38% and 60%. Moreover, we consider a random offset between the codes, test all these cases on a dataset density of 22% and report the average latency over 1,000 iterations. We observe that CTC-PND outperforms Diff-codes with 38% and 60% duty-cycle by 79% and 58% on average, respectively.

Next, we compare CTC-PND against Diff-codes in the symmetric case. To this end, we use duty-cycles of 5%, 10%, 38% and 60% in both networks. Again, we consider a random offset between the codes and report the average discovery latencies from 1,000 iterations. Fig. 4 shows the corresponding results as achieved by Diff-codes with different duty-cycles as well as CTC-PND with different number of SYN nodes. Particularly, Fig. 4a and Fig. 4b present the average discovery latency for dataset densities of 1% and 55%, respectively. As expected, higher duty-cycles lead to lower discovery latencies when any number of SYN nodes are used. Moreover, the mean discovery latency corresponds to the probability of two overlapping slots in addition to the probability of nodes being in coverage area. At high densities such as 55%, where most nodes are in coverage area, all Diff-codes converge to a fixed discovery latency value since the dominant factor is the probability of overlapping active slots. Therefore, increasing the number of SYN nodes does not affect the discovery latency on average. Only high duty-cycles decrease the latency, showing that the most efficient duty-cycle is 100% as in CTC-PND.

Overall, Fig. 4 demonstrates that CTC-PND achieves average discovery latencies that are 100 times lower than Diff-codes with a 5% duty-cycle under all evaluated scenarios.

C. Energy balance in the set cover problem

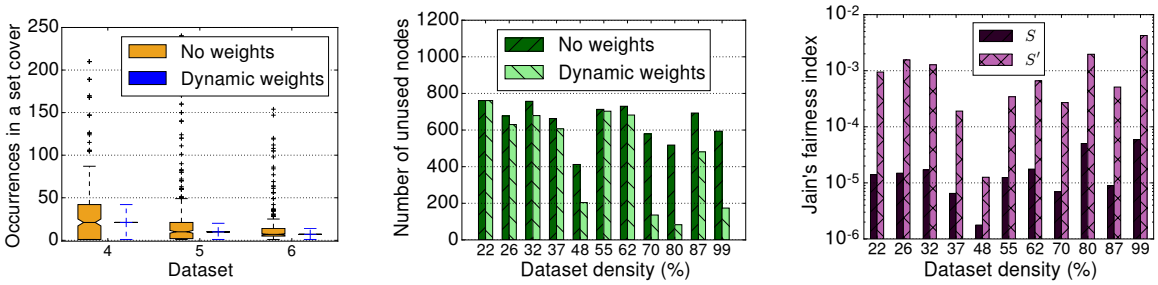
We evaluate the effectiveness of the proposed approach with dynamic weights (DW) by generating 210 set covers based on the datasets 4, 5 and 6 (originally proposed in [23]) from the OR-Library, a collection of test datasets widely used in operation research. Fig. 5a shows the number of times each node appears in a set cover for the schemes with no weights and with dynamic weights; the figure does not include nodes that do not belong to any set cover. Indeed, The DW approach evenly distributes the nodes in the set covers.

Fig. 5b shows the number of unused nodes with and without DWs for the synthetic datasets described in Section VI-A. Clearly, unused nodes with DW are always less than or equal to those without DW. For some of the considered cases, these number gets close to zero. Therefore, the use of dynamic weights in the set cover problem improves the fair use of nodes and, consequently, balances the energy utilization in the network.

We also use the same synthetic datasets to evaluate the performance of Alg. 2. In particular, we calculate the Jain's fairness index (the higher the better) under both a cyclical schedule S and an improved cyclical schedule S'. Fig. 5c shows the corresponding results. The figure highlights that density does not affect the difference between the Jain's index of the schedules. However, datasets with high densities (around 99%) obtain much higher fairness. This is explained by the fact that almost any node can be used as CTC receiver in a dense dataset. Therefore, set covers are made of a few nodes, which makes it easier to find a schedule with few repetitions of nodes. On the other hand, a cyclical schedule that uses all set covers, such as S, is more likely to contain an uneven distribution of nodes. In detail, S' achieves a Jain's index which is around 97% higher on the average, for all densities. Although the improvement is significant, S' is not the optimal solution to this problem since it is obtained with a greedy algorithm. Moreover, a perfectly fair scenario (i.e., a Jain's index of one), is only possible in cases where the intersection of any pair of set covers is empty, e.g., for dataset densities of 100%.

VII. Conclusion

This work introduced a CTC channel establishment scheme that leverages all nodes in co-located networks to carry CTC messages. By doing so, we aimed at decreasing discovery latencies and achieving fair energy consumption among the nodes. Extensive simulations showed that the proposed CTC network discovery achieves a discovery latency that is up to 100 times lower than the state of the art,



(a) (b) (c)

Fig. 5: (a) Occurrence of nodes in set covers, (b) number of unused nodes, and (c) Jain's fairness index.

while effectively balancing energy consumption among the nodes. As future work, we seek to further optimize the energy distribution among the nodes by finding better heuristics and optimization algorithms.

Acknowledgments

This work was partially supported by the Academy of Finland under grants number 299222 and 319710. The authors would like to thank Nidia Obscura Acosta for her help in formalizing the problem as a set cover.

References

[1] W. Jiang, S. M. Kim, Z. Li, and T. He, "Achieving receiver-side cross-technology communication with cross-decoding," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18, 2018, pp. 639–652.

[2] Z. Li and T. He, "Webee: Physical-layer cross-technology communication via emulation," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17, 2017.

[3] W. Jiang, Z. Yin, R. Liu, Z. Li, S. M. Kim, and T. He, "Bluebee: A 10,000x faster cross-technology communication via phy emulation," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17, 2017.

[4] Z. Yin, W. Jiang, S. M. Kim, and T. He, "C-morse: Cross-technology communication with transparent morse coding," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[5] W. Wang, S. He, L. Sun, T. Jiang, and Q. Zhang, "Cross-technology communications for heterogeneous iot devices through artificial doppler shifts," *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 796–806, 2019.

[6] W. Jiang, Z. Yin, S. M. Kim, and T. He, "Transparent cross-technology communication over data traffic," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[7] X. Guo, Y. He, X. Zheng, L. Yu, and O. Gnawali, "Zigfi: Harnessing channel state information for cross-technology communication," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 360–368.

[8] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, May 2009.

[9] R. Pozza, M. Nati, S. Georgoulas, K. Moessner, and A. Gluhak, "Neighbor discovery for opportunistic networking in internet of things scenarios: A survey," *IEEE Access*, vol. 3, pp. 1101–1131, 2015.

[10] M. J. McGlynn and S. A. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, ser. MobiHoc '01, 2001, pp. 137–145.

[11] N. Karowski, A. C. Viana, and A. Wolisz, "Optimized asynchronous multi-channel neighbor discovery," in *Proceedings IEEE INFOCOM*, April 2011, pp. 536–540.

[12] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, ser. MobiHoc '03, 2003, pp. 35–45.

[13] T. Meng, F. Wu, and G. Chen, "Code-based neighbor discovery protocols in mobile wireless networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 806–819, Apr. 2016.

[14] S. Chen, A. Russell, R. Jin, Y. Qin, B. Wang, and S. Vasudevan, "Asynchronous neighbor discovery on duty-cycled mobile devices: Integer and non-integer schedules," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '15. New York, NY, USA: ACM, 2015, pp. 47–56.

[15] L. Wei, B. Zhou, X. Ma, D. Chen, J. Zhang, J. Peng, Q. Luo, L. Sun, D. Li, and L. Chen, "Lightning: A high-efficient neighbor discovery protocol for low duty cycle wsns," *IEEE Communications Letters*, vol. 20, no. 5, pp. 966–969, May 2016.

[16] M. Olbrich, A. Zubow, S. Zehl, and A. Wolisz, "Wiplus: Towards lte-u interference detection, assessment and mitigation in 802.11 networks," in *European Wireless 2017; 23th European Wireless Conference*, May 2017, pp. 1–8.

[17] S. Rayanchu, A. Patro, and S. Banerjee, "Airshark: Detecting non-wifi rf devices using commodity wifi hardware," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 137–154.

[18] ——, "Catching whales and minnows using wifinet: Deconstructing non-wifi interference using wifi hardware," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 57–70.

[19] A. Hithnawi, H. Shafagh, and S. Duquennoy, "Tiim: Technology-independent interference mitigation for low-power wireless networks," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. New York, NY, USA: ACM, 2015, pp. 1–12.

[20] M. Chwalisz and A. Wolisz, "Towards efficient coexistence of IEEE 802.15.4e TSCH and IEEE 802.11," in *IEEE/IFIP NOMS*, 2018.

[21] A. Mahmood, R. Exel, and T. Bigler, "On clock synchronization over wireless LAN using timing advertisement mechanism and TSF timers," in *IEEE ISPCS*, 2014.

[22] Z. Ajami and S. Cohen, "Enumerating minimal weight set covers," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, April 2019, pp. 518–529.

[23] J. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, no. 1, pp. 85 – 93, 1987.

[24] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Annals of Operations Research*, vol. 98, no. 1, pp. 353–371, Dec 2000.

[25] E. K. Baker, "Efficient heuristic algorithms for the weighted set covering problem," *Computers & Operations Research*, vol. 8, no. 4, pp. 303 – 310, 1981.