2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 979-8-3503-2433-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASS58611.2023.00050

ECA: Exclusive Cell Allocation for Autonomous Scheduling in Time-Slotted Channel Hopping

Juhun Shin*, Hongchan Kim*, Jeongyeup Paek†, Saewoong Bahk*

*Department of Electrical and Computer Engineering and INMC, Seoul National University, Seoul, Republic of Korea
†Department of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea
jhshin@netlab.snu.ac.kr, hckim@netlab.snu.ac.kr, jpaek@cau.ac.kr, sbahk@snu.ac.kr

***Abstract*—*Time-slotted channel hopping* (TSCH) is a synchronous MAC protocol designed to meet the requirements of many Internet of Things systems and applications. TSCH requires a scheduling scheme to construct a rendezvous schedule between a communication pair, and *autonomous scheduling* is the state-of-the-art approach where each node schedules autonomously without extra control overhead. However, autonomous scheduling is oblivious of the schedules of other nodes due to its autonomous and thus near-sighted characteristic, and this gives rise to the *cell conflict problem* where multiple nodes attempt to communicate with a single node simultaneously, leading to communication failures and network performance degradation. To this end, we proposed *ECA*, an *exclusive cell allocation* scheme for autonomous scheduling in TSCH. *ECA* utilizes the parent–child relations in routing information to construct a *local regulation* that ensures exclusive cell allocation without additional control messages. Moreover, *ECA* is compatible with existing autonomous scheduling schemes such as ALICE and A3. We implement *ECA* on real embedded devices using ContikiOS, and evaluate on a 71-node testbed to confirm that *ECA* improves link-layer packet reception ratio by 39.8%, latency by 86.3%, and end-to-end packet delivery ratio by 22.9%.**

***Index Terms*—time-slotted channel hopping (TSCH), autonomous scheduling, cell allocation, Internet of Things**

I. Introduction

Time-slotted channel hopping (TSCH) in IEEE 802.15.4e [1] is a MAC protocol designed to connect energy-constrained low-power devices in *Internet of Things* (IoT) systems and applications such as smart city [2], [3], environment monitoring [4], health care [5], and smart building [6]. TSCH's synchronous communication allows for low duty cycle and thus low energy consumption, and its channel hopping enables robust and reliable communication by mitigating interference and collisions (§II-A). However, while TSCH offers a basic framework for synchronized communication and channel hopping, it does not specify how to create, maintain, or modify the schedule for each node to communicate effectively. For this reason, numerous TSCH scheduling schemes have been proposed in the literature to address the scheduling challenges in different network environments and scenarios.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A5A1027646 & No. 2022R1A4A5034130), and also by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-2021-0-02048) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

There are generally three main categories of TSCH scheduling: centralized, distributed, and autonomous (§II-B). The topic of this paper is on *autonomous scheduling*, the state-of-the-art approach where each node independently constructs its own schedule without requiring extra control messages nor negotiation. In autonomous scheduling, each node self-determines its schedule without considering, and oblivious to, the schedules of other nodes. This near-sighted characteristic of autonomous scheduling leads to the *cell conflict problem* where multiple nodes may attempt to communicate with a single node simultaneously without being aware of the conflict (§II-D). The *cell conflict problem* causes communication failures due to collisions, and thus has a negative impact on both network performance and node energy efficiency. However, none of the existing autonomous scheduling schemes could completely eliminate the root cause of the problem.

To address this issue, we propose *ECA*, an *exclusive cell allocation* scheme for autonomous scheduling in TSCH (§IV). Autonomous scheduling typically relies on routing information such as next hop node ID, and *ECA* utilizes the parent–child relations to generate a virtual node ID called *local index* by piggybacking a small amount of information to the routing messages that will be exchanged anyway. Using the *local index*, *ECA* resolves the *cell conflict problem* by allocating cells exclusively through a rule called *local regulation* that is shared among all TSCH nodes. Moreover, *ECA* is designed to be compatible with existing autonomous scheduling schemes such as ALICE and A3 (§III).

We implement *ECA* on real embedded devices (M3 boards) using ContikiOS, and evaluate on a 71-node public testbed (§V). The results show that *ECA* improves link-layer packet reception ratio by 39.8%, latency by 86.3%, and end-to-end packet delivery ratio by 22.9%.

The contributions of this work are as follows:

- We propose *ECA* to address *cell conflict problem* with novel *local index* and *local regulation* design.
- We design *ECA* to be compatible with existing autonomous scheduling schemes.
- We implement and evaluate *ECA* on a large-scale testbed to demonstrate a significant improvement in PDR, latency, and duty cycle performance.

The remainder of this paper is organized as follows: §II presents background and motivation of this work, and §III

2155-6814/23/$31.00 ©2023 IEEE
DOI 10.1109/MASS58611.2023.00050

discusses the related work. *ECA* is proposed in §IV, and §V present evaluation results. §VI provides the limitation and future work. Finally, §VII concludes the paper.

II. Background and Motivation

We first provide a brief overview of TSCH and RPL routing protocol, and then describe the *cell conflict problem* that this work aims to address.

A. IEEE 802.15.4e TSCH

TSCH [1] is a synchronous MAC protocol that adopts time-slotted communication and channel-hopping techniques.

Time-slotted communication: To establish synchronization, TSCH nodes periodically broadcast Enhanced Beacon (EB), which carries the time as well as other information about the TSCH network. A node attempting to join a TSCH network listens to this EB, and synchronizes to the time information in EB. Fig. 1 illustrates an example of TSCH operation. TSCH divides time into timeslots. A *timeslot* in TSCH is typically 10 ms long, which is sufficient for exchanging a single frame and its acknowledgement (ACK). A *slotframe* is a collection of timeslots repeated continuously over time, and it provides a common reference for all nodes to synchronize their communication. The number of timeslots in a slotframe is called *slotframe size* (L_{SF}), which determines the repetition period of a slotframe. For example, in Fig. 1, slotframe size is 3 so the slotframe repeats every 3 timeslots. *Absolute slot number* (ASN) is defined as the total number of timeslots that has passed since the beginning of the TSCH network. *Time offset* (t_{o}) is a relative position within a slotframe, and it is calculated as,

$$t_{\mathrm{o}} = mod(ASN,\ L_{\mathrm{SF}}). \tag{1}$$

Channel-hopping: TSCH nodes switch communication channel every timeslot to avoid collision and alleviate multipath fading through frequency diversity. The channel to be used in a specific ASN is calculated as,

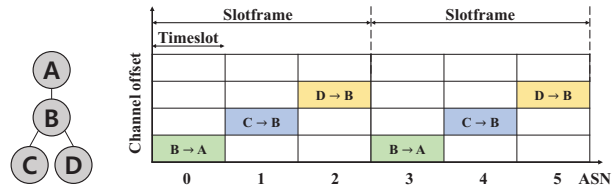
$$Channel = List_c[mod(ASN + c_{\mathrm{o}},\ N_{List_c})]. \tag{2}$$

where $List_c$ is a set of channels designated for hopping and N_{List_c} is the number of elements in $List_c$. By means of *channel offset* (c_{o}), it is possible to exploit different frequency channels in the same timeslot.

However, TSCH standard does not define how each node determines its own time and channel offsets (t_{o}, c_{o}), and the process of determining these values are left to the implementation. In other words, the TSCH standard decouples the definition of scheduling from the main standard to provide a high degree of flexibility, and this led to numerous TSCH scheduling schemes being suggested in the literature [7]–[14].

B. TSCH scheduling

There are three main categories of TSCH scheduling that can synchronize the operation of nodes in a TSCH network: centralized, distributed, and autonomous scheduling.



(a) Topology (b) Slotframe schedule

Fig. 1. An example of TSCH operation.

Centralized scheduling: A capable central node constructs the schedule of all nodes in a TSCH network [7], [8]. To do this, the central node must collect information about all nodes in the network, and then distributes the schedule back to all nodes. Although this approach may potentially provide an optimal solution, it generates a large number of control messages and cannot respond rapidly to topology changes.

Distributed scheduling: Nodes construct their schedule through continuous negotiation with each other without requiring a central node [9], [10], [14]. This results in relatively fewer control messages and faster adaptation to topology changes compared to centralized scheduling.

Autonomous scheduling: Each node autonomously self-determines its own schedule [11]–[13]. To build transmission and reception schedules without control messages, autonomous scheduling typically relies on routing information and utilizes a shared rule (e.g., pseudo-random hash function) across all nodes in the network.

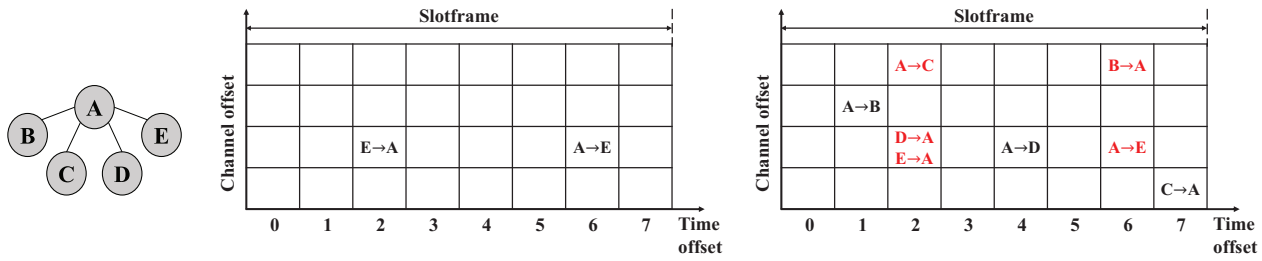
In ALICE [13], for example, the time offset for the link from node A to node B can be calculated by using the following equation.

$$t_{\mathrm{o}} = mod(Hash(\alpha \cdot ID(A) + ID(B)),\ L_{\mathrm{SF}}). \tag{3}$$

The coefficient α is employed to distinguish traffic direction, and $ID(x)$ is the node ID of node x. Since a node can obtain the node ID of its parent or child in the routing messages while establishing a parent-child relationship in the routing layer, the TSCH network is no longer afflicted by control message overhead. However, as a consequence of each node autonomously scheduling itself, autonomous scheduling, by nature, lacks the ability to take the schedules of other nodes into consideration.

C. RPL

RPL is the IETF standard *IPv6 routing protocol for low-power and lossy network*, designed for resource-constrained IoT devices [15]–[19]. RPL is a distance-vector type protocol that builds a tree-like routing topology called *destination-oriented directed acyclic graph* (DODAG). Each node broadcasts *DODAG information object* (DIO) messages to exchange and update routing information. Upon reception of a valid DIO message, each node selects a neighboring node as its *parent node* which is its upward route toward the root. After selecting a parent, a node then sends a *destination advertisement object* (DAO) message to the selected parent in order to inform it



(a) Example topology (b) Slotframe schedule from perspective of node E (c) Overall slotframe schedule

Fig. 2. Illustration of an example cell conflict problem with one parent, four children nodes, and slotframe size of 8.

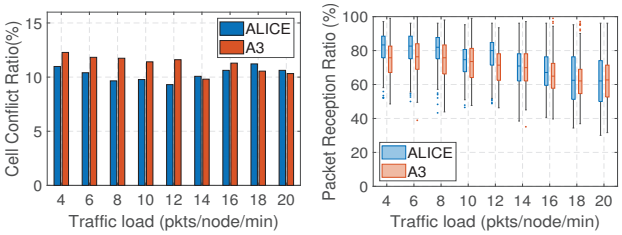
of the downward path. If a node needs to change its upward route by selecting a new parent, it sends a DAO message to the new parent to establish the new upward route, and also sends a *no-path DAO* message to its previous parent to evict the downward route from the previous parent.

D. The cell conflict problem

The task of a TSCH scheduling scheme is to decide time offset (t_o) and channel offset (c_o) for each communication pair (sender and receiver). A tuple of $\{t_o, c_o\}$ is called a *cell*. Autonomous scheduling is myopic since it does not consider the schedules of other nodes in its cell allocation process. This results in *cell conflict* where multiple nodes attempt to communicate with a single node simultaneously.

For example, in a TSCH network as in Fig. 2(a), node E would only know the schedule between itself and its parent node as shown in Fig. 2(b). With autonomous scheduling, the existence of other nodes, such as B, C, and D, are unknown, and their schedules are also unobservable. This near-sighted characteristic leads to a conflicting overall schedule as shown in Fig. 2(c). When time offset is 2, three nodes (A, D, and E) try to communicate with node A. However, node A can only execute one cell at a time. If node A selects the cell A→C, it will be unable to receive transmissions of other cells. Therefore, nodes E and D would need to retransmit in the next slotframe. Furthermore, even if node A selects either of the cell D→A or E→A, there is a high probability of collision since node D and E will transmit to A at the same time (t_o) on the same channel (c_o). Note that the *cell conflict problem* can occur even when the channel offset is different, as long as the time offset is the same.

The *cell conflict problem* can result in unnecessary collision or interference because nodes transmit packets that their intended receivers cannot receive. Moreover, if the packet cannot be transmitted successfully in a particular slotframe due to *cell conflict*, it needs to be retransmitted in the next slotframe, which increases channel usage airtime as well as energy consumption which has a negative impact on the overall network performance. Additionally, in heavy traffic scenarios with a large number of nodes or higher data rate, *cell conflict* can occur more often and lead to severe network performance degradation. Although attempts have been made to avoid this issue by utilizing the randomness of hash functions [13] or



(a) Cell conflict ratio (b) Packet reception ratio

Fig. 3. Cell conflict ratio and link-layer packet reception ratio of ALICE and A3 autonomous schedulers from 71-node testbed experiment

by allocating the channel offset in node-based manner [20], there is always a probability of *cell conflict* due to the modular operation in Eq. (1). Therefore, neither are fundamental solutions to the problem. Our preliminary experiments with ALICE and A3 autonomous schedulers (details in §V) reveal a significant occurrence of the *cell conflict*, resulting in a poor link-layer performance as shown in Fig. 3. This work aims to fundamentally solve this problem.

III. Related Work

Orchestra [11] is the first autonomous scheduling scheme for TSCH in which each node constructs its own schedule locally and autonomously by leveraging information from the routing layer. Orchestra has inspired several follow-up research, resulting in a number of autonomous scheduling schemes being proposed.

ALICE [13] discloses the limitations of node-based autonomous scheduling, Orchestra. ALICE constructs its schedules autonomously based on *directional links* instead of on a per-node basis. However, multiple directional link-based scheduling leads to collision and contention problems, including *cell conflict*. To address these issues, a time-varying scheduling technique is adopted. Using the randomness of a hash function, nodes reschedule after each slotframe so that the schedule is not fixed and prevents reoccurring collisions or contention in subsequent slotframes. Nevertheless, the time-varying scheduling merely decreases the chance of reoccurrence; it is unable to eliminate the root cause of the problem

A3 [12] is an adaptive autonomous cell allocation scheme where each node adapts its transmission and reception slots in

response to varying traffic loads. A3 divides the slotframe into multiple zones, allowing for the allocation of multiple tx/rx slots within a single slotframe. Each node occupies a small number of slots for energy efficiency in low-traffic situations, and assigns additional slots when traffic is heavy. To enable autonomous operation with these adaptive characteristics, each node continuously self-estimates the traffic load based on the transmission queue for each receiver at the sender side and by observing the transmitter's operation on the receiver side. However, the *cell conflict problem* impairs the performance of A3 by interfering with the receiver-side observation of the transmitter's operation. Additionally, A3 divides the slotframe into multiple zones to assign slots, which reduces the slotframe size and leads to a more frequent occurrence of *cell conflict problem*.

There are other alternative/hybrid approaches as well. For example, OST [14] manages communication slots adaptively for each directional link to respond to traffic demand in a *distributed* manner. Leveraging its distributed nature, OST employs a binary resource tree to prevent *cell conflict problem*. DualBlock [21] addresses a similar but different issue. It solves the problem of multiple senders being unable to sense each other's transmissions due to aligned clear channel assessment (CCA) operation when transmitting in the same timeslot. However, no approach has been proposed that eliminates the root cause of the *cell conflict problem* in autonomous scheduling.

IV. ECA Design

We propose *ECA*, an exclusive cell allocation scheme for TSCH autonomous schedulers to overcome the *cell conflict problem*. This section presents its design.

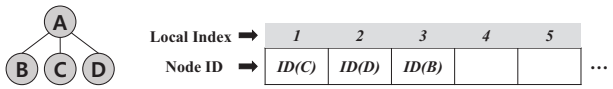
A. Overview

ECA aims to reduce the occurrences of *cell conflict* between parent and children nodes by leveraging the concepts of *local index* and *local regulation*. By introducing the *local index*, which is allocated to the children nodes by the parent node, *ECA* can get far-sighted information for scheduling (e.g., the existence of other children nodes or the time offset of their schedules). Then, while adhering to the principles of autonomous scheduling, *local regulation* utilizes the *local index* to enable exclusive resource allocation. We note that *ECA* is orthogonal to, and compatible with, existing autonomous scheduling schemes such as ALICE and A3; that is, *ECA* applies to any autonomous TSCH schedulers to handle the *cell conflict problem*.

B. Local index

Here we first introduce the rationale behind the concept of *local index* in *ECA*, and highlight its advantages in cell scheduling.

Traditional autonomous scheduling schemes mainly rely on routing information. For example, schedulers like Orchestra and ALICE utilize the node IDs of neighboring nodes in the routing layer (i.e., parent and children nodes) to self-determine schedules for communicating with them. However, relying solely on routing information has limitations in solving the *cell conflict problem*. Although the routing information does provide the node IDs of parent and children nodes, it does not provide the node IDs of sibling nodes. Therefore, autonomous scheduling is not only unable to consider the schedules of sibling nodes, it may not even be aware of their existence. This forces near-sighted scheduling and prevents proactive handling of the *cell conflict problem*. To this end, *ECA* incorporates *local index*.



(a) Topology (b) Local indices of node A

Fig. 4. An example of *local index* assignment.

Local index enables far-sighted scheduling by allowing each node to be aware of its sibling nodes. In the context of RPL's tree-based routing relationships, the parent node is responsible for assigning the *local index* to its children nodes. When a child node selects a parent (next hop towards the routing root), it sends a DAO message to notify the parent of its existence. Upon receiving the DAO message and identifying the sender as a new child node, the parent node chooses a *local index* to assign to the child node. The parent then piggybacks the selected *local index* in the link-layer ACK for the DAO message [1]. This procedure allows for the swift distribution of *local indices* without additional overhead.

Local index starts from one and increases sequentially as the parent node assigns it to children nodes in the order of joining. If a child node with a specific *local index* departs from the parent node (chooses a different parent as a result of RPL's parent selection process), that *local index* simply becomes empty; the *local indices* for other remaining children nodes do not change. When a new child node joins the parent node and such empty *local indices* exists, those empty *local indices* will be filled in first since the parent assigns any unused *local index* in increasing order from the beginning. Once the parent node gives a specific *local index* to a child, it does not change until the child node leaves its parent.

Fig. 4 illustrates an example of *local index* assignment. We assume a topology with four nodes as shown in Fig. 4(a), where node A is the parent and the other nodes are its children. The process of assigning *local index* is depicted in Fig. 4(b) using an array. Assuming that nodes C, D, and B join node A in that order, they are assigned *local indices* of 1, 2, and 3, respectively. If node D changes its parent and leaves node A, *local index* 2 becomes empty, and the *local indices* of the remaining nodes remain as is. Then, if another node, say E, selects node A as its parent, node A would assign *local index* 2 to node E, filling the gap.

[1]The device may include additional content in a link-layer ACK encapsulated as IEs

Due to the way *local indices* are assigned, each node is able to infer the existence of sibling nodes with smaller *local indices*. For example, in Fig. 4, since node D is assigned a *local index* of 2, it can infer that there must be a node with *local index* 1 even without knowing the real identity of that node. Similarly, node B with *local index* of 3 is aware of the existence of nodes with *local indices* 1 and 2. The key idea is that a node needs to know only the existence and *local indices* of the sibling nodes, **not their real node IDs**, since *ECA* will be scheduling based on the *local indices*. Based on this awareness, each node can perform scheduling that takes into account its sibling nodes.

Moreover, nodes do not need to worry about nodes with larger *local indices* since those nodes are responsible for considering nodes with smaller *local indices*. For example, node D is aware of the existence of a node with *local index* 1 (C) but not a node with *local index* 3 (B). However, since node B is aware of the existence of nodes with *local indices* 1 and 2 (C and D), B can take D into account during the scheduling process, thereby achieving comprehensive consideration among all sibling nodes (i.e., far-sighted scheduling).

C. Local regulation

We propose a new scheduling rule called *local regulation* to prevent the *cell conflict problem* among sibling nodes. *local regulation* is shared and applied across the network, enabling exclusive cell scheduling.

For cell scheduling, *local regulation* utilizes the *local index* of children nodes as input for calculating the time offsets for each directional link between a parent and children instead of solely relying on the node IDs. Specifically, the time offsets for upward link (from a child node n to the parent node P) and downward link (from P to n) are calculated as follows,

$$t_{\mathrm{o}}^{\mathrm{up}} = mod(Hash(\alpha \cdot LI(n) + ID(P)), L_{\mathrm{SF}}) \quad (4)$$

$$t_{\mathrm{o}}^{\mathrm{down}} = mod(Hash(\alpha \cdot ID(P) + LI(n)), L_{\mathrm{SF}}) \quad (5)$$

where *ID* and *LI* denote the node ID and *local index* respectively. It is worth noting that the parent node's ID is known to all children nodes. By using the *local index* instead of the node ID, each node can calculate the schedules of other sibling nodes.

Based on Eqs. (4) and (5), a node determines its own time offset exclusively by executing the CELL CONFLICT AVOIDANCE algorithm in Alg. 1. The occurrences of *cell conflict* depends on having the same time offset, irrespective of the difference in channel offset. Therefore, the CELL CONFLICT AVOIDANCE algorithm focuses on ensuring non-overlapping time offsets. The algorithm takes the parent node *P*'s ID (i.e., *ID(P)*), and the *local index* of node *n* that executes the algorithm (i.e., *LI(n)*) as inputs. The output of the algorithm is the exclusively allocated time offsets of upward and downward links for node *n*.

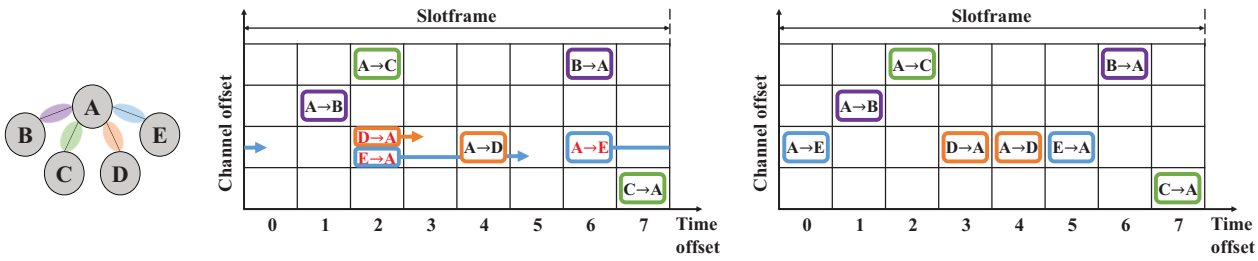
The algorithm first defines a set of time offsets denoted as $\mathcal{T}$ for indicating whether a specific time offset has been allocated or not, and initializes it as an empty set. It then proceeds to calculate the time offsets for each node's upward and downward links, starting from a *local index* of one and iterating up to *LI(n)* (Line 3). In the course of calculating the time offsets, it is possible for the derived time offset to overlap with the schedule of nodes with smaller local indices (e.g., 1, 2, ..., $LI(n)$-1) that were added to set $\mathcal{T}$ earlier. When such a *cell conflict* occurs, the algorithm cyclically shifts the time offset by one until a non-overlapping time offset is found (Line 7), and applies it as the new time offset for the node with local index i (Line 10). If all timeslots are already occupied (Line 11), it becomes physically impossible to allocate a cell exclusively, regardless of the allocation method. In that case, the algorithm terminates (Line 12), and the output is calculated by Eqs. (4) and (5). This exclusive cell allocation process continues until the algorithm reaches the local index *LI(n)* (Line 15).

Algorithm 1 CELL CONFLICT AVOIDANCE

Input: $LI(n)$ and $ID(P)$
Output: $t_{\mathrm{o}}^{\mathrm{up}}(n)$ and $t_{\mathrm{o}}^{\mathrm{down}}(n)$ ▷ time offsets for node n
Initialize:
1: $\mathcal{T} \leftarrow \{\}$ ▷ set of the occupied time offsets
2: $L_{\mathrm{SF}} \leftarrow$ slotframe size
Procedure: exclusive cell allocation
3: **for** $i = 1$ to $LI(n)$ **do**
4: $u = t_{\mathrm{o}}^{\mathrm{up}}(i, ID(P))$ ▷ Eq. (4)
5: $d = t_{\mathrm{o}}^{\mathrm{down}}(i, ID(P))$ ▷ Eq. (5)
6: **for** x *in* $\{u, d\}$ **do**
7: **while** $x \in \mathcal{T}$ **do**
8: $x = \mathrm{mod}(x + 1, L_{SF})$ ▷ cyclic shift
9: **end while**
10: $\mathcal{T} \cup \{x\}$
11: **if** $n(\mathcal{T}) == L_{SF}$ **then**
12: **terminate**
13: **end if**
14: **end for**
15: **end for**
16: $\{t_{\mathrm{o}}^{\mathrm{up}}(n), t_{\mathrm{o}}^{\mathrm{down}}(n)\} = \{u, d\}$

The algorithm is designed such that node *n* can recognize the schedules of nodes with smaller *local indices* and allocate its schedule exclusively. There may be other sibling nodes with *local index* greater than *LI(n)*, but this does not pose any problem because nodes with higher *local index* than *LI(n)* will avoid the schedule of the node with *LI(n)*. Furthermore, note that since the parent node is aware of all *local indices* of the children nodes, the parent can always synchronize its schedule with the children.

An Example: Fig. 5 presents an example of *ECA* scheduling on the RPL topology shown in Fig. 5(a). Let's assume that nodes B, C, D, and E each have *local index* of 1, 2, 3, and 4 in alphabetical order. To make it easier to explain without loss of generality, we assume that t_{o} values determined based on local index (Eqs. (4) and (5)) are the same as those determined based on node ID as illustrated in Fig. 2(c). Then, Fig. 5(b) illustrates how each node avoids *cell conflict* through *local*



(a) Topology (b) Cell conflict avoidance of each node (c) Final slotframe schedule

Fig. 5. An example of *ECA* slotframe schedule with slotframe size 8

regulation. Since node B has a *local index* of 1, it is unaware of the existence of other sibling nodes and therefore determines its t_o to be 1 and 6 without exclusive cell allocation. Having a *local index* of 2, node C becomes cognizant of the presence of node B with a *local index* of 1, allowing it to avoid *cell conflict* through *local regulation* by recognizing that node B's schedule has t_o 1 and 6. However, since the schedules of node B and node C do not overlap, avoidance is not triggered. Node D becomes aware of the existence of nodes B and C and their t_o of 1, 2, 6, and 7 through *local regulation*. Because the t_o of D→A overlaps with t_o of A→C, node D avoids this *cell conflict* by assigning t_o of 3 by shifting by 1. Lastly, node E with the highest *local index* becomes aware of the existence of all sibling nodes, and also infers through *local regulation* that node D occupies t_o of 3 instead of 2 to avoid *cell conflict*. Then, node E allocates its t_o to be 0 and 5 exclusively in the same manner.

If we compare the two final schedules Fig. 5(c) and Fig. 2(c), despite being on the same topology, there are five schedules that experience *cell conflict* in Fig. 2(c) while all schedules are exclusively assigned in Fig. 5(c). The example shows that *ECA* can successfully allocate exclusive cell schedule to all nodes in the network through the use of *local index* and *local regulation*.

V. Evaluation

In this section, we evaluate the efficacy of *ECA* against the state-of-the-art autonomous schedulers on a large-scale testbed. We use ALICE [13] and ALICE with A3 [12] (referred to as A3 in the rest of this paper) as the baseline schedulers.

A. Implementation and experimental setup

We implement *ECA* on M3 boards using Contiki-NG [22], and utilize the publicly available implementations of ALICE[2] and A3[3], which are also implemented in Contiki-NG. We conduct all experiments on FIT/IoT-LAB [23], a large-scale open public testbed using 71 M3 nodes. The transmission power is set to -17 dBm. We use four IEEE 802.15.4 channels (15, 20, 25, 26) for TSCH channel hopping. TSCH timeslot length is set to 10 ms, and the slotframe sizes for EB and broadcast are set to 397 and 17, respectively [12]. The unicast slotframe size is set to 19 for ALICE and 20 for A3 [4].

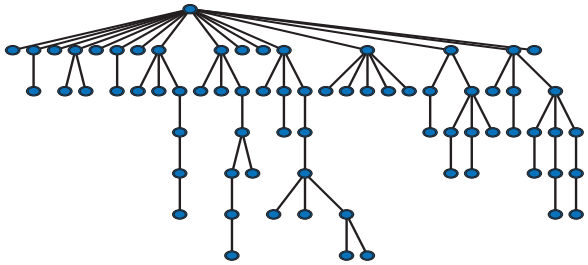


Fig. 6. RPL routing topology snapshot with 71 nodes in FIT/IoT-LAB testbed

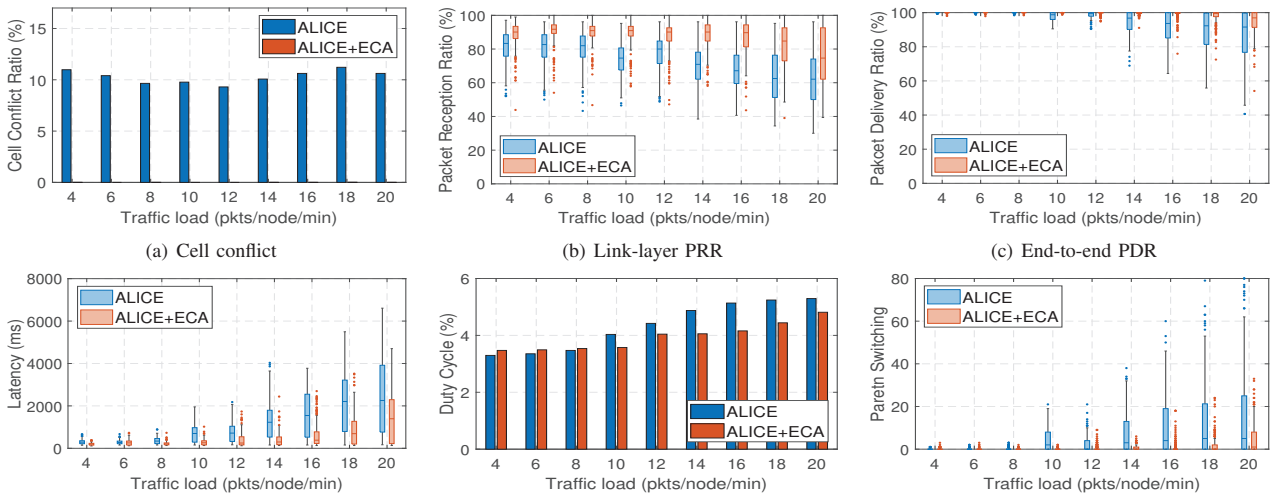
We use RPL in storing mode as the routing layer, and MRHOF [24] with ETX as the objective function for RPL. We employed the RPL and TSCH implementations provided by Contiki-NG, but made a minor fix to prevent nodes from receiving additional packets when their link-layer packet queue is full. This modification enables us to analyze more accurately how *ECA* improves communication performance on each link by reducing packet losses due to queue overflow and their incorrect impact on ETX estimation. During the experiment, RPL dynamically adjusts the network topology, i.e., each RPL node may change its parent. Fig. 6 displays a snapshot of an RPL topology instance in such a setting.

Given this setup, we consider a data collection application scenario where each node transmits data packets to the root node periodically. While changing the upward traffic load from each node to the root node, we compare the performance of ALICE and A3 with and without *ECA*. Data transmission lasted 40 minutes in each experiment, and sufficient time was given for initialization and bootstrap before starting data transmission. For each experimental case, we conducted three repeated experiments.

Performance metrics used for evaluation are as follows:

- **Cell conflict ratio (CCR)** of a node is the ratio of scheduled cells for its children that experience *cell conflict*. Given that a parent node knows all schedules of its children nodes, the parent can calculate its own CCR.

[2]https://github.com/skimskimskim/ALICE
[3]https://github.com/skimskimskim/A3
[4]ALICE requires a prime number, and A3 requires a non-prime number.



(a) Cell conflict (b) Link-layer PRR (c) End-to-end PDR

(d) Per-hop latency (e) Duty cycle (f) Parent switching

Fig. 7. Performance of ALICE and *ECA* with ALICE according to traffic load

- **Link-layer packet reception ratio (PRR)** is the PRR between a node and its parent. Each node except for the root has its own link PRR.
- **End-to-end packet delivery ratio (PDR)** is the PDR of data packets from each node to the root.
- **Per-hop latency** is calculated by dividing the end-to-end latency of each data packet by the number of hops that the packet passes.
- **Duty cycle** is the ratio of time the radio was turned on during the entire duration of the experiment.
- **Parent switching** is the number of times a node switches its parent in the RPL routing layer. This is influenced by link quality, and has a significant impact on the end-to-end PDR.

B. Comparison with ALICE

Fig. 7 presents the experiment results on ALICE, with and without *ECA*, with varying traffic load.

Average *cell conflict ratios* in Fig. 7(a) show that *cell conflicts* in ALICE have been successfully eliminated by *ECA* from ∼10% to 0% (red bars are not shown because the values are zeros). The 10% CCR with ALICE not only means that 10% of link communication may experience failure, but also that retransmissions are necessary in the event of failure. Retransmissions require additional resources in the next slotframe, causing a cascading effect and leading to serious network performance degradation. However, ALICE with *ECA* never experiences any *cell conflict* regardless of the traffic load. This indicates that *ECA* is highly effective in eliminating the *cell conflicts*.

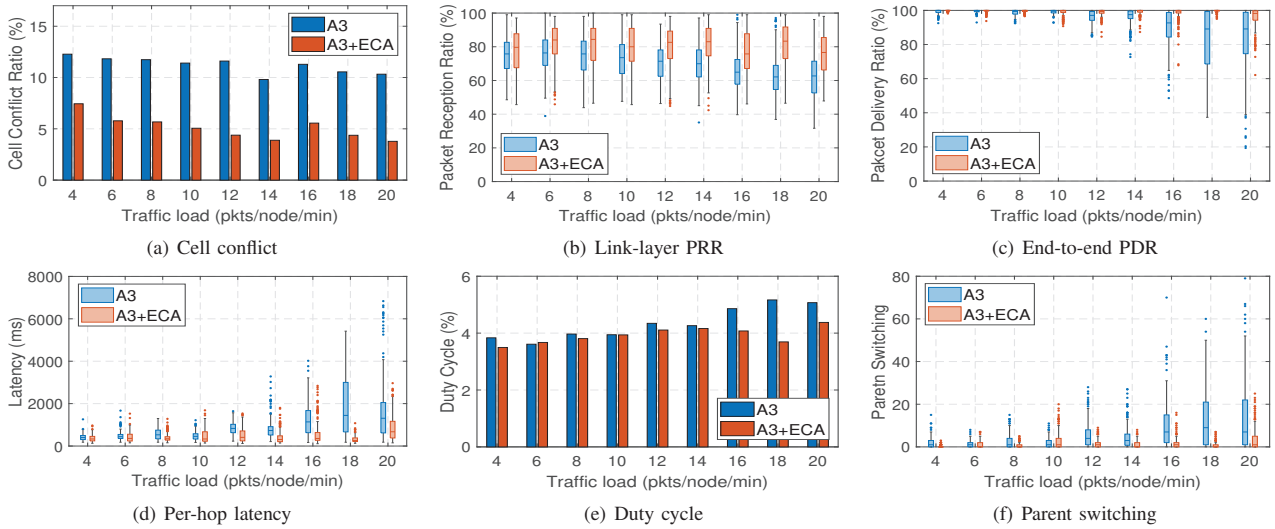
Eliminating *cell conflict* has a direct impact on improving the link-layer PRR as shown in Fig. 7(b). Note that although traffic load has no impact on the schedule itself (Fig. 7(a)), it does have an impact on other network performance metrics (Figs. 7(b) to 7(f)), such as link PRR, due to additional contention, interference, and congestion.

The improvement in performance at the individual link layer leads to significant improvements in higher-level performance metrics across the entire network, such as end-to-end PDR (Fig. 7(c)), per-hop latency (Fig. 7(d)), duty cycle (Fig. 7(e)), and routing layer stability (Fig. 7(f)). Furthermore, the performance gains increase as the traffic load increases. This is because, at low traffic load, some of the scheduled cells are unused, and thus a cell conflict does not necessarily result in a collision. This is the reason why the PDR in Fig. 7(c) is close to 100% at low traffic load despite the presence of *cell conflicts*[5]. However, as the traffic load increases, the probability of transmissions taking place in overlapped schedules also increases, and thus a *cell conflict* has a higher chance of resulting in a collision. Therefore, the impact of *cell conflict* is larger under high traffic load, and the gain of *ECA* is further emphasized.

Fig. 7(d) shows the improvement in per-hop latency, where the gain is notably larger than that of PDR. This is because retransmissions can mask some of the *cell conflicts* from the PDR perspective, but every retransmission counts towards latency.

An increase in retransmission also leads to an increase in the radio duty cycle with a similar trend to latency as shown in Fig. 7(e); *ECA* improves duty-cycle at higher traffic load. However, under a very low traffic load, the duty cycle of ALICE seems slightly superior. This is because *ECA* inherently raises the baseline of the duty cycle by allocating schedules *exclusively* to prevent *cell conflict*. In other words, because schedules are evenly distributed throughout the slotframe without overlap, more cells are allocated overall. Recall that when a cell is scheduled, the receiver side radio must

[5]The reason why end-to-end PDR is higher than link-layer PRR despite using one or more links is due to link-layer retransmissions.



(a) Cell conflict (b) Link-layer PRR (c) End-to-end PDR

(d) Per-hop latency (e) Duty cycle (f) Parent switching

Fig. 8. Performance of A3 and *ECA* with A3 according to traffic load

be turned on to listen to potential incoming packets. In the case of ALICE, *cell conflicts* mean a smaller number of active cells due to overlap in schedule, which we regard as incorrect behavior. We believe *ECA*'s allocation without overlap is the correct behavior, and the loss in the duty cycle at low traffic is insignificant.

Through Fig. 7(f), we observe the effect of *ECA* on the stability of RPL routing topology. The *cell conflict* occurs due to schedule overlap, regardless of the link quality, but it causes a fluctuation in RPL's perceived link quality by deteriorating the link-layer PRR. As a result of the fluctuation in link quality, the node decides to switch its parent node to another node with better link quality. Thus, *cell conflict* leads to the increment of parent switching. *ECA* eliminate unnecessary parent switching by mitigating fluctuation in link quality caused by *cell conflict*.

C. Comparison with A3

Fig. 8 presents the experiment results on A3, with and without *ECA*, with varying traffic load. By dividing the slotframe into multiple zones, A3 can control the allocation of Tx/Rx slots. The slotframe size of A3 is 20, with 4 zones utilized, resulting in each zone having a slotframe size of 5.

The impact of *ECA* to mitigate *cell conflict* is shown in Fig. 8(a). In contrast to the comparison with ALICE, *ECA* cannot entirely eliminate the *cell conflicts* because A3 has a slotframe size of 5 for each zone. If more than five children nodes are assigned to a single parent node, it becomes impossible to exclusively allocate all schedules regardless of how the schedules are arranged since the slotframe size is limited to 5. Despite this, *ECA* distributes schedules uniformly to minimize *cell conflicts* to the extent possible, and provides fully exclusive schedules when the number of children nodes is lower than or equal to the slotframe size of each zone. As a result, *ECA* reduces *cell conflicts* to less than half in all traffic load scenarios. Accordingly, the improvement in link-layer PRR resulting from *cell conflict* reduction can be observed in Fig. 8(b) with a max average gain of 39.8% at 18 pkts/node/min.

The reduced slotframe size for each zone causes a higher incidence of *cell conflicts*. Also, *cell conflict* has a negative impact on A3's self-monitoring of traffic for autonomous traffic adaptation. If a *cell conflict* occurs when traffic increases, the transmitter will increase the Tx slot regardless of the presence of *cell conflict* by looking at the transmission queue. On the other hand, the receiver has to infer the traffic coming toward it by observing the transmitter's operation. If the receiver experiences a *cell conflict*, it will miss the corresponding packet and thus may not be aware of whether the transmitter has actually sent the packet or not. As a result, the receiver will maintain its Rx slot without making any changes. *Cell conflict* causes a discrepancy in autonomous slot assignment between the transmitter and receiver nodes, leading to A3 being unable to respond adequately to traffic. Consequently, the *cell conflict* has a negative impact on the performance of A3, coupled with A3's characteristic of employing smaller zones and autonomous adaptation to traffic.

Through the mitigation of *cell conflict* (Fig. 8(a)), A3 with *ECA* can enjoy improved link quality and more properly adjust its Tx/Rx slot allocation to match the current traffic load (Fig. 8(b)). Therefore, there is a remarkable improvement in all network performance aspects (Figs. 8(c) to 8(f)). In general, all network performance results exhibit similarities and share the same trends as with ALICE in §V-B. Specifically, Fig. 8(c) and Fig. 8(d) shows improved end-to-end PDR and latency with a max average gain of 22.9% and 86.3%, respectively, at 18 pkts/node/min. Also, both duty cycle (Fig. 8(e)) and parent switching (Fig. 8(f)) show improvements similar to those seen in §V-B.

VI. Limitation and Future Work

Here we discuss the limitations of *ECA* and future work.

Limitation: *ECA* effectively solves the *cell conflict problem* among sibling nodes by using the parent–child relations in routing information. However, there is a possibility of *cell conflict* between the schedule of a parent node to its parent (i.e. grandparent) and its children, and *ECA* does not eliminate this conflict. *ECA* chose this design because of the following reason: If *ECA* mandates the elimination of such conflict, then all the children nodes have to change their schedule whenever the parent changes its parent (i.e. grandparent) because the schedule between the parent and grandparent node has been changed. Eventually, every node below the node that had changed its parent should adjust its schedule recursively, leading to a negative impact on network stability. Therefore, only considering the schedule between sibling nodes is a practical decision with better expected average performance.

Future work: *ECA* uses a virtual node ID called *local index* to resolve only the *cell conflict problem*. But the fact that the node ID is virtual can also be used for traffic adaptation of autonomous scheduling. By monitoring the amount of traffic towards a node, the node can receive an additional *local index* through the parent node's DIO message, enabling it to adjust the number of cell allocations based on the traffic load. We leave this as our future work.

VII. Summary and Conclusion

The self-determination feature of the current autonomous scheduling schemes results in the exclusion of any consideration of other nodes' schedules. This feature gives rise to *cell conflict problem* where multiple nodes attempt to communicate with a single node simultaneously, resulting in a significant loss in network performance. In this paper, we proposed exclusive cell allocation for autonomous scheduling to address *cell conflict problem* without additional control messages. By means of schedule-aware *local regulation* that effectively utilizes routing information, *ECA* eliminates *cell conflict* across all nodes in a TSCH network. Also, *ECA* is compatible with autonomous scheduling methods such as ALICE and A3. Through a large-scale testbed of 71 nodes, we demonstrated that *ECA* enables reliable and low-latency communication even in heavy traffic load.

References

[1] R. Heile, R. Alfvin, P. Kinney, J. Gilb, and C. Chaplin, "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer," IEEE, Tech. Rep., 2012.

[2] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2018, pp. 188–199.

[3] H.-S. Kim, H. Cho, M.-S. Lee, J. Paek, J. Ko, and S. Bahk, "MarketNet: An Asymmetric Transmission Power-based Wireless System for Managing e-Price Tags in Markets," in *The 13th ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2015.

[4] S. A. Malek, F. Avanzi, K. Brun-Laguna, T. Maurer, C. A. Oroza, P. C. Hartsough, T. Watteyne, and S. D. Glaser, "Real-Time Alpine Measurement System Using Wireless Sensor Networks," *Sensors*, vol. 17, no. 11, p. 2583, 2017.

[5] J. Park, W. Nam, T. Kim, J. Choi, S. Lee, D. Yoon, J. Paek, and J. Ko, "Glasses for the Third Eye: Improving Clinical Data Analysis with Motion Sensor-based Filtering," in *The 15th ACM International Conference on Embedded Networked Sensor Systems (SenSys'17)*, Nov. 2017, pp. 8:1–8:14.

[6] K. Brun-Laguna, A. L. Diedrichs, D. Dujovne, C. Taffernaberry, R. Leone, X. Vilajosana, and T. Watteyne, "Using SmartMesh IP in smart agriculture and smart building applications," *Computer Communications*, vol. 121, pp. 83–90, 2018.

[7] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop IEEE 802.15. 4e networks," in *IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC)*, 2012, pp. 327–332.

[8] R. Soua, P. Minet, and E. Livolant, "MODESA: An optimized multi-channel slot assignment for raw data convergecast in wireless sensor networks," in *IEEE 31st international performance computing and communications conference (IPCCC)*, 2012, pp. 91–100.

[9] S. Jeong, J. Paek, H.-S. Kim, and S. Bahk, "TESLA: Traffic-aware elastic slotframe adjustment in TSCH networks," *IEEE Access*, vol. 7, pp. 130 468–130 483, 2019.

[10] J. Jung, D. Kim, T. Lee, J. Kang, N. Ahn, and Y. Yi, "Distributed slot scheduling for QoS guarantee over TSCH-based IoT networks via adaptive parameterization," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2020, pp. 97–108.

[11] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proceedings of the 13th ACM conference on embedded networked sensor systems (SenSys)*, 2015, pp. 337–350.

[12] S. Kim, H.-S. Kim, and C.-k. Kim, "A3: Adaptive autonomous allocation of TSCH slots," in *The 20th IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2021, pp. 299–314.

[13] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *The 18th IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019, pp. 121–132.

[14] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-Demand TSCH Scheduling with Traffic-awareness," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, July 2020, pp. 69–78.

[15] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6550

[16] H.-S. Kim, J. Ko, D. E. Culler, and J. Paek, "Challenging the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2502–2525, 2017.

[17] H.-S. Kim, J. Paek, and S. Bahk, "RPLIE: RPL for indoor environments under midterm link fluctuations," *Journal of Communications and Networks*, vol. 23, no. 3, pp. 201–211, 2021.

[18] H. Kim, H.-S. Kim, and S. Bahk, "MobiRPL: Adaptive, robust, and RSSI-based mobile routing in low power and lossy networks," *Journal of Communications and Networks*, vol. 24, no. 3, pp. 365–383, 2022.

[19] M. Park, G. Jeong, O. Gnawali, and J. Paek, "RPL Objective Function for Multihop PLC Network," *Journal of Communications and Networks*, vol. 25, no. 1, pp. 131–139, Feb. 2023.

[20] A. Elsts, S. Kim, H.-S. Kim, and C. Kim, "An empirical survey of autonomous scheduling methods for TSCH," *IEEE Access*, vol. 8, pp. 67 147–67 165, 2020.

[21] J. Park, H. Kim, H.-S. Kim, and S. Bahk, "DualBlock: Adaptive Intra-Slot CSMA/CA for TSCH," *IEEE Access*, vol. 10, pp. 68 819–68 833, 2022.

[22] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *IEEE international conference on local computer networks*, 2004, pp. 455–462.

[23] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 459–464.

[24] O. Gnawali and P. Levis, "The Minimum Rank with Hysteresis Objective Function," RFC 6719, Sep. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6719