HRL-TSCH: A Hierarchical Reinforcement Learning-Based TSCH Scheduler for IIoT

F. Fernando Jurado-Lasso, *Member, IEEE*, Charalampos Orfanidis, *Member, IEEE*, J. F. Jurado, and Xenofon Fafoutis, *Senior Member, IEEE*

***Abstract*—The Industrial Internet of Things (IIoT) demands adaptable Networked Embedded Systems (NES) for optimal performance. Combined with recent advances in Artificial Intelligence (AI), tailored solutions can be developed to meet specific application requirements. This study introduces HRL-TSCH, an approach rooted in Hierarchical Reinforcement Learning (HRL), to devise Time Slotted Channel Hopping (TSCH) schedules provisioning IIoT demand. HRL-TSCH employs dual policies: one at a higher level for TSCH schedule link management, and another at a lower level for timeslot and channel assignments. The proposed RL agents address a multi-objective problem, optimizing throughput, power efficiency, and network delay based on predefined application requirements. Simulation experiments demonstrate HRL-TSCH's superiority over existing state-of-art approaches, effectively achieving an optimal balance between throughput, power consumption, and delay, thereby enhancing IIoT network performance.**

***Index Terms*—Industrial Internet of Things (IIoT), Networked Embedded Systems (NES), sleep scheduling, Time Slotted Channel Hopping (TSCH), Reinforcement Learning (RL), Software-Defined Wireless Sensor Networks (SDWSNs).**

I. Introduction

THE INTERNET of Things (IoT) is a transformative technology that interconnects objects to the Internet, paving the way for innovative applications and services [1]. These objects equipped with sensors, processing capabilities, power sources, and wireless communication radios, form the backbone of the Industrial Internet of Things (IIoT). The IIoT specifically leverages IoT devices to facilitate real-time control and monitoring of industrial processes, enhancing operational efficiency and decision-making [2].

Manuscript received 18 January 2024; revised 7 May 2024; accepted 28 May 2024. Date of publication 3 June 2024; date of current version 6 December 2024. Manuscript received 18 January 2024; revised 7 May 2024; accepted 28 May 2024. This work was partly supported by DAIS. DAIS (https://dais-project.eu/) has received funding from the ECSEL Joint Undertaking (JU) under Grant 101007273. The JU was supported in part by the European Union's Horizon 2020 research and innovation programme and Sweden, Spain, Portugal, Belgium, Germany, Slovenia, Czech Republic, Netherlands, Denmark, Norway, and Turkey. Danish participants were supported in part by the Innovation Fund Denmark under Grant 0228-00004A. The associate editor coordinating the review of this article and approving it for publication was C. W. Tan. *(Corresponding author: F. Fernando Jurado-Lasso.)*

F. Fernando Jurado-Lasso, Charalampos Orfanidis, and Xenofon Fafoutis are with the Embedded Systems Engineering Section, DTU Compute, Technical University of Denmark, 2800 Lyngby, Denmark (e-mail: ffjla@dtu.dk; chaorf@dtu.dk; xefa@dtu.dk).

J. F. Jurado is with the Department of Basic Science, Faculty of Engineering and Administration, Universidad Nacional de Colombia Sede Palmira, Palmira 763531, Colombia (e-mail: jfjurado@unal.edu.co).

Digital Object Identifier 10.1109/TCCN.2024.3408459

In the realm of IIoT applications, diverse needs arise, ranging from data rate and delay to throughput and power usage [3]. Some applications demand energy-efficient modes to extend the lifespan of sensor nodes, particularly in remote and challenging environments. Others prioritize low latency for real-time industrial control, while some strike a fair balance between reliability and power efficiency. As operational needs evolve, the network must dynamically adapt to ensure seamless performance.

Networked Embedded Systems (NES) traditionally serve as the foundation for both IoT and IIoT. These networks consist of numerous low-cost, low-power wireless sensor nodes (often called Sensor Nodes (SNs)) deployed in the environment to monitor and control physical phenomena [4]. In challenging environments susceptible to interference and multipath fading, a conventional approach involves utilizing Time Slotted Channel Hopping (TSCH), a Media Access Control (MAC) protocol of the IEEE 802.15.4 standard. TSCH schedules packet transmission and reception in a time-slotted fashion, overcoming the challenges of interference and multipath fading [5].

The effectiveness of a TSCH network heavily relies on its scheduler, responsible for assigning timeslots and channels to communication links. While redundant links enhance network reliability and minimize packet delay, they may also increase power consumption. Therefore, the TSCH scheduler must be cautiously designed to meet the requirements of IIoT applications, ensuring flexibility and adaptability to dynamic changes.

In this context, this paper introduces Hierarchical Reinforcement Learning for Time-Slotted Channel Hopping (HRL-TSCH). This innovative approach, grounded in Hierarchical Reinforcement Learning (HRL), is intricately designed to customize TSCH schedules according to the specific requirements of IIoT applications. The hierarchical framework comprises RL agents responsible for learning the optimal schedule and a TSCH selection algorithm enabling SNs to efficiently select the nearest scheduled link for a designated destination address. The incorporation of HRL proves highly beneficial as it efficiently manages the inherent complexity of the TSCH link scheduling problem. By partitioning decision-making into higher and lower levels, our approach optimizes network performance by considering both global changes and local link-specific policies. This hierarchical structure significantly enhances adaptability, efficiency in exploration and exploitation, scalability, and generalization capabilities. The proposed methodology contributes to the

field by providing a flexible and responsive solution that maximizes network performance in dynamic IIoT environments. In brief terms, the contributions of this paper are as follows:

1) We develop a HRL architecture specifically designed for solving the TSCH scheduler problem in IIoT networks that aims to maximize network performance.
2) We formulate of a comprehensive mathematical model for estimating power consumption, delay, and throughput within a TSCH network. This model serves as the foundation for the RL agent's learning process to derive the optimal schedule.
3) We develop a TSCH selection algorithm for end devices to efficiently select the nearest scheduled link corresponding to a specific destination address.
4) We conduct a comprehensive performance evaluation of the proposed approach using various IIoT application requirements within the Cooja network simulator.

The rest of the paper is organized as follows. Section II provides an introduction to key concepts and technologies relevant to this study. In Section III, we review related work, focusing on centralized and decentralized schedulers. We highlight the limitations of existing centralized approaches in adapting to dynamic changes, underscoring the need for flexibility. This informs our analysis and the development of our proposed framework. Section IV offers a comprehensive overview of the methodology employed in designing the TSCH scheduler. Section V focuses on the RL approach utilized in the design of the TSCH scheduler. Section VI presents the performance evaluation of the proposed approach, showcasing its effectiveness and efficiency. Lastly, Section VII concludes the paper and provides insights into future directions and potential avenues for further research.

II. Background

A. Time Slotted Channel Hopping (TSCH)

The TSCH protocol operates at the link-layer and is characterized by two key features: time synchronization of nodes and frequency hopping functions [5]. These components work in tandem to facilitate *frequency-division multiple access* and *time-division multiple access*, enabling multiple network nodes to efficiently share the same radio medium by dividing its available bandwidth into frequency sub-channels.

Allocation of access in discrete timeslots within each sub-channel frequency is managed to ensure equitable sharing among nodes. In centralized schedulers, a designated *coordinator* node assumes the responsibility of generating a *schedule* based on these rules. This schedule is then disseminated to all nodes in the network.

To join the TSCH network, every SN must receive and adhere to this schedule, which defines the *slotframe* (i.e., the set of timeslots) and the *channel hopping sequence* (i.e., the set of frequency sub-channels) to be used by the network. In essence, the TSCH schedule comprises a set of links, specifying the actions each node must take at designated timeslots.

SNs within the network have the option of receiving or transmitting packets or entering a sleep state to conserve energy. The organization of nodes' transmissions and receptions in the schedule, along with the slotframe size, are critical parameters that significantly influence the performance of the TSCH network. These factors impact throughput, delay, and energy efficiency.

B. Reinforcement Learning (RL)

RL is a branch of ML that excels at solving complex problems across various domains like robotics and games [6]. In RL, an agent learns the best policy π by interacting with the environment, aiming to maximize the cumulative reward $\mathcal{R}$ [7]. At each time step t, the agent takes action $a_t \in \mathcal{A}$, and the environment responds with a reward $r_{t+1} \in \mathcal{R}$ and a new state $s_{t+1} \in \mathcal{S}$. The agent's goal is to learn the optimal policy π that maximizes the cumulative reward $\mathcal{R}$ [8]. There are two main approaches to RL: value-based and policy-based. In value-based RL, the agent learns the optimal policy by estimating the value function $V(s)$ or $Q(s, a)$. In policy-based RL, the agent learns the optimal policy directly without estimating the value function. In this paper, we use the Deep Q-Learning (DQN) algorithm, a value-based RL algorithm, to train the RL agents.

1) Deep Q-Network (DQN): Since our HRL framework is a model-free approach, we use the DQN algorithm to train the RL agents. The DQN algorithm is a variant of the Q-learning algorithm that uses a deep neural network to approximate the action-value function $Q(s, a)$, and a replay buffer to store the experiences and then sample from it to train the agent. Q-learning updates the state-action value function $Q(s, a)$ through an iterative process. It employs a Bellman equation as the basis for its updates. The value of $Q(s, a) \leftarrow (1 - \sigma)Q(s, a) + \sigma[r + \lambda \max_{a' \in \mathcal{A}} Q(s', a')]$ is updated by taking a weighted sum of the previous Q-value and new information obtained from the current state and action. Where $\sigma \in [0, 1]$ and $\lambda \in [0, 1]$ are the learning rate and discount factor, respectively. s' and a' are the next state and action, respectively. r is the immediate reward of the policy π.

DQN approximates the Q-value function $Q(s, a)$ using a deep neural network such that $Q(s, a) \approx Q(s, a; \theta)$, where θ is the set of parameters of the neural network. The neural network is trained to minimize the loss function $L(\theta) = \mathbb{E}_{\tau \sim U(.)}[(\phi - Q(s, a; \theta))^2]$, where $\phi = r + \lambda \max_{a' \in \mathcal{A}} Q(s', a'; \theta^-)$ Where τ is the experience tuple (s, a, r, s'), and $U(.)$ is the uniform distribution over all possible experience tuples. θ^- is the set of parameters of the target network. The target network is a copy of the neural network that is used to calculate the target Q-value.

C. Hierarchical Reinforcement Learning (HRL)

HRL is a branch of RL that aims to solve complex problems by decomposing them into smaller sub-problems [9]. In HRL, there are two levels of policies: a higher-level policy π_h and a lower-level policy π_l. The higher-level policy π_h selects the sub-goal g that the agent should achieve. The lower-level policy π_l selects the action a that the agent should take to

achieve the sub-goal g. The higher-level policy π_h is trained using the lower-level policy π_l. The lower-level policy π_l is trained using the reward function of the higher-level policy π_h. In this paper, we use the HRL framework to design the TSCH scheduler. The higher-level policy π_h selects the link that the agent should schedule. The lower-level policy π_l selects the timeslot and channel that the agent should assign to the selected link. For further insights into HRL, readers can refer to [10]. Additionally, those interested in applications of ML in NES can explore [11], [12].

D. TSCH and Reinforcement Learning (RL)

In the realm of RL, the TSCH schedule can be linked to a *policy* dictating the actions of network nodes. This policy serves as a mapping function, correlating the current state of the network with the appropriate action to undertake. The network state encompasses variables such as the current timeslot, channel, and the status of network links. Within the framework of RL, the coordinator assumes the role of an *agent*, tasked with learning the optimal policy through interactions with the environment.

III. Related Work

TSCH schedulers fall into four categories: centralized, decentralized, static, and hybrid [14], [28]. In this section, we focus on centralized and decentralized schedulers, given their relevance to our work.

A. Centralized Schedulers

Centralized schedulers are designed by a scheduling algorithm that runs on a central entity, such as a controller or a border router, that has a global view of the network. For instance, in [13] the authors propose a scheduling algorithm that reserves sequential slots along the path from the source to the destination, minimizing packet delay in multihop TSCH networks. However, the algorithm complexity increases with the network size. Dynamic scheduling is addressed in [14], which proposes an algorithm considering end-to-end delay, network throughput, and Packet Delivery Ratio (PDR) to handle mobility in TSCH networks. Despite its effectiveness, tracking node mobility introduces significant overhead, and dynamic adaptation to the traffic pattern is not considered. A cross-layer approach optimizing topology and TSCH schedule is presented in [15] to minimize latency. While achieving high timeslot utilization, power consumption remains a concern. Reference [16] proposes a scheduling algorithm aiming to maximize network throughput and ensure max-min fairness. However, prioritizing throughput may lead to increased power consumption. Reference [17] adopts a RL-based approach to optimize slotframe size in TSCH networks. While slotframe size is optimized based on specific application requirements, the TSCH schedule itself is not optimized. Optimizing the Minimal Scheduling Mechanism using RL and Markov Decision Process (MDP) is proposed by Nguyen-Duy et al. [18], where nodes keep their radios activated based on traffic patterns from various application scenarios. However, dynamic adaptation of the schedule is not considered. Reference [19] evaluates nine Multi-Armed Bandit (MAB) algorithms to select the optimal channel and introduces a mechanism integrating selected algorithms with TSCH to improve reliability and energy efficiency. Nevertheless, the mechanism lacks dynamic adaptation to traffic patterns. Other noteworthy centralized schedulers include [20], introducing a scheduling algorithm rooted in Software-Defined Networking (SDN) for centralized scheduling in TSCH networks, and [21], proposing a scheduling algorithm using a whitelist to avoid collisions.

B. Decentralized Schedulers

The decentralized schedulers include autonomous approaches that allow SNs to autonomously select the timeslots and channels to transmit and receive packets and collaborative approaches that require SNs to collaborate to design the schedule. An example is presented in [22], where a Deep Reinforcement Learning (DRL) framework is employed to offer Quality of Service (QoS) features. However, this framework lacks awareness of changes in application requirements and primarily focuses on the parameter configuration of the TSCH protocol. In [23], QL-TSCH is proposed, an autonomous scheduling algorithm that uses a multi-agent Q-learning approach to optimize slot allocation in TSCH networks. Each agent is responsible for a node and learns the optimal transmission slot based on the network conditions. The algorithm uses an action peeking mechanism, which allows nodes to keep track of the actions of their neighbors by constantly monitoring the time slots, to reduce learning time and improve convergence rate. However, the algorithm suffers from high energy consumption due to constant monitoring of the time slots. In [24], an autonomous scheduling algorithm is introduced, allowing each node to construct its schedule without negotiation overhead. Nevertheless, the Orchestra schedule may lead to suboptimal network performance due to its lack of consideration for network conditions. A traffic-aware scheduling algorithm for 6TiSCH networks is proposed in [25]. This algorithm utilizes cell allocation information to enhance load balancing and improve bandwidth utilization. However, it lacks the flexibility to enable run-time reconfiguration of the schedule. Addressing emergency scenarios, [26] presents an emergency-aware scheduling algorithm for TSCH networks. This algorithm prioritizes emergency traffic with high reliability and bounded delay. While effective, it hijacks cells allocated to existing traffic flows when an emergency is detected. In the realm of reinforcement learning, Bommisetty and Venkatesh [27] introduce a Phasic Policy Gradient (PPG) based TSCH schedule learning algorithm. This algorithm formulates the scheduling policy as an optimization problem, outperforming totally distributed and totally centralized DRL-based scheduling algorithms through the use of the actor-critic policy gradient method.

In summary, HRL-TSCH distinguishes itself from the surveyed approaches in several key aspects, as illustrated in Table I that compares the surveyed approaches based on their architecture, ML approach, dynamic requirements support,

TABLE I
SUMMARY AND COMPARISON OF RELATED WORK

Scheduler	Architecture	ML	Req.	Dyn. schedule	Optimization	key features
[13]	Centralized	✗	✗	✗	TSCH schedule	Sequential slot reservation algorithm that minimizes the packet delay in multihop TSCH networks.
[14]	Centralized	✗	✗	✓	TSCH schedule	Dynamic scheduling algorithm for mobility in TSCH networks.
[15]	Centralized	✗	✗	✗	TSCH schedule	High timeslot utilization algorithm that minimizes the latency.
[16]	Centralized	✗	✗	✗	TSCH schedule	Throughput and max-min fairness scheduling algorithm.
[17]	Centralized	RL	✓	✓	TSCH Slotframe size	RL approach that optimizes the slotframe size of TSCH networks.
[18]	Centralized	RL	✗	✗	TSCH schedule	RL approach that optimizes the Minimal Scheduling Mechanism.
[19]	Centralized	RL	✗	✗	TSCH schedule	RL approach that optimizes the channel selection.
[20]	Centralized	✗	✗	✓	TSCH schedule	A Proof-of-concept SDN-based scheduling algorithm to enable centralized scheduling in TSCH networks.
[21]	Centralized	✗	✗	✗	TSCH schedule	A scheduling algorithm that uses a whitelist to avoid collisions.
[22]	Decentralized	DRL	✗	✗	TSCH parameters	DRL framework that configures the parameters of TSCH networks.
[23]	Decentralized	RL	✗	✓	TSCH schedule	Multi-agent Q-learning approach that optimizes slot allocation in TSCH networks.
[24]	Decentralized	✗	✗	✗	TSCH schedule	Autonomous scheduling algorithm that builds its own schedule.
[25]	Decentralized	✗	✗	✗	TSCH schedule	Traffic-aware scheduling algorithm that facilitates load balancing.
[26]	Decentralized	✗	✗	✗	TSCH schedule	Emergency-aware scheduling algorithm that forwards emergency traffic with high reliability and bounded delay.
[27]	Decentralized	✗	✗	✗	TSCH schedule	DRL-based scheduling algorithm that implements the scheduling policy as an optimization problem.
HRL-TSCH	**Centralized**	**HRL**	✓	✓	**TSCH schedule**	**HRL framework that optimizes the TSCH schedule based on the application requirements.**

dynamic schedule support, optimization target, and key features.

1) *Focus on Schedule Design:* While previous works, such as [17], [22], primarily concentrate on the parameter configuration of the TSCH protocol, HRL-TSCH takes a unique approach by placing its emphasis on the actual design of the TSCH schedule.
2) *Dual Policies:* In contrast to surveyed approaches [17], [18], [19], [22] that predominantly use a single policy, HRL-TSCH introduces a dual-policy framework. This dual-policy approach enhances the adaptability and optimization capabilities of the TSCH schedule.
3) *Awareness of Application Changes (Adaptability):* HRL-TSCH is designed to be aware of changes in application requirements, dynamically adjusting the TSCH schedule in real-time to maximize network performance based on evolving needs. This adaptability is facilitated through two key mechanisms: Firstly, by adopting an SDN-based architecture, HRL-TSCH enables swift, real-time modifications to the underlying TSCH schedule as necessitated by shifting user demands. Secondly, through the utilization of RL in the policy formulation, the HRL agent continuously seeks the optimal TSCH schedule based on current user requirements, thus enabling adaptive optimization in response to changing conditions. This capability sets HRL-TSCH apart from surveyed approaches, which generally lack awareness of application changes and dynamic adaptation, except for [17], which optimizes the slotframe size.

To the best of our knowledge, HRL-TSCH marks the first attempt to introduce an HRL approach dedicated to optimizing TSCH performance through tailored schedule design, tailored to meet the unique requirements of each application scenario. For readers interested in a more comprehensive survey of TSCH schedulers, a detailed overview can be found in [14], [28].

IV. SYSTEM DESIGN

This section presents an overview of the methods and techniques employed in this study to address the TSCH scheduling problem in IIoT. By delving into the intricacies of the methodology, readers will gain a deeper understanding of how the research tackles the problem at hand and how the proposed solution is designed and evaluated. It also serves as a roadmap for understanding the core components and processes involved in the research, providing insight into the system architecture, network model, and mathematical formulation of performance metrics. The notation used throughout the paper is summarized in Table II.

A. System Architecture

In this paper, we adopted the three-tier network architecture principles proposed in [17] to design a RL-based scheduler that is aware of changes in the application requirements and adapts the TSCH schedule accordingly to maximize the network performance. The overall architecture is shown in Fig. 1 and consists of three planes: the application, control, and data

TABLE II
NOTATION

Symbol	Description
$\mathcal{N}$	Set of nodes in the network
$\mathcal{E}$	Set of links between nodes
$\lvert\mathcal{E}\rvert$	Number of links in the network
$\mathcal{W}$	Set of forwarding paths
$\mathcal{H}$	Set of TSCH slots in the schedule
$\mathcal{U}$	Set of timeslots in a TSCH schedule
$\lvert\mathcal{U}\rvert$	Number of timeslots in a TSCH schedule
$\lvert\mathcal{U}_{n,tx}\rvert$	Number of transmitting timeslots of node n
$\mathcal{Z}$	Set of channels in a TSCH schedule
$\lvert\mathcal{Z}\rvert$	Number of channels in a TSCH schedule
$F_{n,m}$	Set of forwarding nodes of node n in the forwarding path m
φ	Set of application requirements
$\lvert u\rvert$	Duration of a timeslot
T	Throughput of the network
$T^{max}_{n,\lvert\mathcal{U}_{n,tx}\rvert}$	Maximum throughput achieved by node n with $\lvert\mathcal{U}_{n,tx}\rvert$ transmitting timeslots
T_0	Traffic in packets per second
$T_{children,n}$	Incoming traffic from the children of node n
B_n	Total traffic generated by node n
$\xi_{T,n}$	Noise term that accounts for the uncertainty in the throughput model of node n
P	Power consumption of the network
P_n	Power consumption of node n
$P_{n,tx}$	Power consumption of node n in the transmitting state
$P_{n,rx}$	Power consumption of node n in the receiving state
E^f_{tx}	Energy consumption of the node n in the transmitting state
E^f_{rx}	Energy consumption of the node n in the receiving state
$E^f_{rx\_ack}$	Energy consumption of the node n in the receiving acknowledgment state
$E^f_{tx\_ack}$	Energy consumption of the node n in the transmitting acknowledgment state
E^f_{listen}	Energy consumption of the node n in the idle listening state
$\lvert\mathcal{H}^{idle}_{rx,n}\rvert$	Number of cells in the TSCH schedule of node n over a time span of one second that are in the receiving state, but no packets are received
P_0	Power consumption of SNs for basic operations
$\xi_{p,n}$	Noise term that accounts for the uncertainty in the power consumption model of node n
D	Worst-case delay of the network
$D(n,f,u)$	Number of timeslots required for a data packet to travel from source node n to forwarding node f using timeslot u
$D(f,m)$	Number of timeslots needed for a data packet to traverse from forwarding node f to the next forwarding hop f_m or the destination node
$D_{f,\mathcal{Q}}$	Delay of packets in the queue of forwarding node f
λ_f	Arrival rate of packets at forwarding node f
μ_f	Service rate of packets at forwarding node f
K	Large constant that represents the delay of packets in the queue of a forwarding node when the system is unstable
$\xi_{d,n}$	Noise term that accounts for the uncertainty in the worst-case delay model of node n
L	Loss rate of the network
$\mathcal{R}$	Set of rewards
$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions
π_h	Higher-level policy
π_l	Lower-level policy
ψ_h	Penalty for the higher-level policy
ψ_{π_l}	Penalty for the lower-level policy

plane. *The application plane* is the entrance point of the system and is responsible for receiving the weights of the application requirements from the user and sending them to

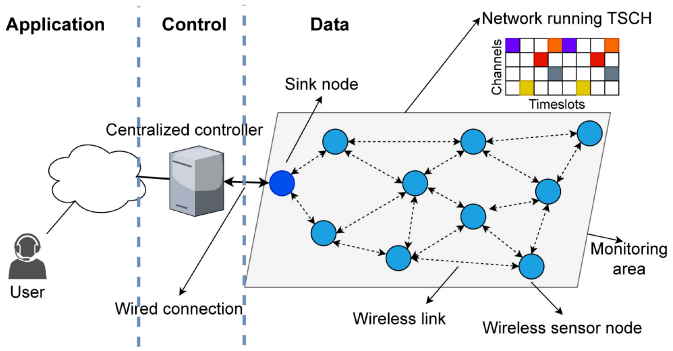

Fig. 1. System architecture.

the control plane. *The control plane* hosts multiple modules for the correct operation of the system including the data collection and the network management module [29]. The network manager module provides an Application Program Interface (API) to reconfigure the network. The control plane also hosts modules for the generation of forwarding paths and TSCH schedules, and the RL agent. Lastly, *the data plane* is the network infrastructure built upon SNs (light blue nodes in Fig. 1) that collect the data of physical phenomena of interest from the *monitoring area* (e.g., industrial plant, agricultural field, etc.) and enforces, at each node, the forwarding paths and the TSCH communication links generated by the RL agent and distributed by the control plane. SNs, in the data plane, exchange packets with the control plane to report their status and receive the new forwarding paths and TSCH schedules using the *sink node* (dark blue node in Fig. 1) as a gateway in a multi-hop fashion. For a more comprehensive understanding of each plane and its modules, refer to [17].

Given that the training of the RL agent occurs offline, constructing a surrogate environment becomes imperative to emulate the characteristics of a real-world TSCH network. Notably, online training within a testbed or network simulator is deemed impractical due to potential slowdown in the training process. In the upcoming section, we delve into the network model and present the mathematical formulation of the performance metrics employed in this study.

B. TSCH Network Model

We model the network as a graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes in the network, and $\mathcal{E}$ is the set of communication links between nodes. Each node $n \in \mathcal{N}$ is associated with a set of attributes, such as the node's position, forwarding paths, TSCH schedule, and traffic load. They can communicate directly with neighboring nodes within its transmission range. Thus, we have defined the set of communication links within the transmission range as:

$$\mathcal{E} = \{(n_x, n_y) \mid n_x, n_y \in N, x \neq y \text{ and node } n_y \text{ is within the transmission range of node } n_x\} \quad (1)$$

In our network model, each node can act as a source, a destination, or a relay node, except for the sink node. A source node generates data packets and sends them toward a destination node, while a destination node receives data packets

from a source node. Additionally, a relay node facilitates data transmission by forwarding packets from the source node to the destination node.

The centralized entity, which is later introduced, that manages the network in this study is called the *controller*. The controller sets the forwarding paths $w \in \mathcal{W}$ and the TSCH slots $h \in \mathcal{H}$ of the network at any given time. The forwarding paths are the paths that data packets follow from a source node to a destination node. The set $\mathcal{H}$ is the TSCH schedule that determines the timeslots $u \in \mathcal{U}$ and channel offsets $\zeta \in \mathcal{Z}$ that nodes use to transmit and receive data packets.

C. Mathematical Formulation of the Performance Metrics

This section presents the mathematical formulation of the performance metrics to optimize in the TSCH scheduler. These metrics also serve as a benchmark for assessing the system's capabilities and comparing it with other scheduling approaches.

1) Throughput: We use the throughput of the network as a measure of the efficiency of the network. We express the throughput of the network (T) as a function of the number of packets delivered to the controller over a period of time. The maximum throughput, in packets per second, achieved by node n given that it has $|\mathcal{U}_{n,tx}|$ transmitting timeslots is calculated as follows:

$$T^{max}_{n,|\mathcal{U}_{n,tx}|} = \frac{|\mathcal{U}_{n,tx}|}{|\mathcal{U}| \times |u|} \tag{2}$$

where $T^{max}_{n,|\mathcal{U}_{n,Tx}|}$ represents the maximum throughput achieved by node n with $|\mathcal{U}_{n,tx}|$ transmitting timeslots. Then, the throughput of the network can be expressed as follows:

$$T_n = \begin{cases} B_n, & \text{if } T_{children,n} < T^{max}_{n,|\mathcal{U}_{n,tx}|} - T_0 \\ T^{max}_{n,|\mathcal{U}_{n,tx}|}, & \text{otherwise} \end{cases} \tag{3}$$

where T_0 is the traffic in packets per second that is generated by each $n \in \mathcal{N}$, this includes control and data packets. $T_{children,n} = \sum_{c \in \mathcal{N}_{child,n}} T_c$, depicts the incoming traffic from the children of node n, and $B_n = T_0 + T_{children,n}$ is the total traffic generated by node n. The throughput of the network is then calculated as follows:

$$T = \frac{\sum_{n \in \mathcal{N}} \left(T_n + \xi_{T,n}\right)}{|\mathcal{N}|} \tag{4}$$

$$\text{subject to} \qquad |\mathcal{U}| > 0,\ |\mathcal{N}| > 0, |u| > 0 \tag{5}$$

where $\xi_{T,n} \sim N(0, \sigma^2_{T,n})$ is a noise term that accounts for the uncertainty in the throughput model of node n.

2) Power Consumption: In SNs, the power consumption (P) is significantly influenced by different radio communication states, such as transmit and receive states [30]. However, in TSCH networks, these states do not have equal contributions to node power consumption. Receiving timeslots is likely to contribute more to the power consumption since SNs activate their radios even when there are no packets to receive. On the other hand, transmitting timeslots is likely to contribute less to the power consumption since SNs only activate their radios when there are packets to transmit [31], [32]. We assume that the power consumption of the network is mainly attributed to the receiving and transmitting states. We can then calculate the power consumption of node n for both states as follows:

$$P_{n,tx} = T_n \times \left(E^f_{tx} + E^f_{rx\_ack}\right) \tag{6}$$

$$P_{n,rx} = T_{children,n} \times \left(E^f_{rx} + E^f_{tx\_ack}\right) + \left|\mathcal{H}^{idle}_{rx,n}\right| \times E^f_{listen} \tag{7}$$

where E^f_{tx}, E^f_{rx}, $E^f_{rx\_ack}$, $E^f_{tx\_ack}$, and E^f_{listen} are the energy consumption of the node n in the transmitting, receiving, receiving acknowledgment, transmitting acknowledgment, and idle listening states, respectively. $|\mathcal{H}^{idle}_{rx,n}| = T^{max}_{n,|\mathcal{U}_{n,Rx}|} - T_{children,n}$ is the number of cells in the TSCH schedule of node n over a time span of one second that are in the receiving state, but no packets are received. Then, $P_n = P_0 + P_{n,tx} + P_{n,rx} + \xi_{p,n}$, where P_0 captures the power consumption of SNs for basic operations, such as neighbor discovery and synchronization, and $\xi_{p,n} \sim N(0, \sigma^2_{p,n})$ is a noise term that accounts for the uncertainty in the power consumption model of node n. Therefore, the network power consumption can be expressed as follows:

$$P = \frac{\sum_{n \in \mathcal{N}} \left(P_n + \xi_{p,n}\right)}{|\mathcal{N}|} \tag{8}$$

$$\text{subject to} \qquad |\mathcal{N}| > 0, E^f_{tx} > 0, E^f_{rx} > 0, E^f_{rx\_ack} > 0, \\ E^f_{tx\_ack} > 0, E^f_{listen} > 0 \tag{9}$$

3) Worst-Case Delay: Our objective is to minimize the worst-case delay (D) in the network, which refers to the delay of the data packet that experiences the longest delivery time. We use the notation $D(n, f, u)$ to denote the number of timeslots required for a data packet to travel from source node n to forwarding node $f \in F_{n,m} \subset \mathcal{W}$ using timeslot $u \in \mathcal{U}$. Similarly, $D(f, m)$ represents the number of timeslots needed for a data packet to traverse from forwarding node f to the next forwarding hop f_m or the destination node. It's important to note that forwarding nodes relay data packets using their closest scheduled link to the next hop. Mathematically, we can express the delay of node n as follows:

$$D_n = \max_{u \in \mathcal{U}} \left(D(n, f, u) + \sum_{f \in F} D(f, f_m) \right) \times |u| + \sum_{f \in F} D_{f,\mathcal{Q}} \tag{10}$$

where $|u|$ represents the duration of a timeslot. $D_{f,\mathcal{Q}}$ is the delay, in milliseconds, of packets in the queue of forwarding node f, and it is calculated as follows:

$$D_{f,\mathcal{Q}} = \begin{cases} \frac{\lambda_f}{\mu_f \times (\mu_f - \lambda_f)} \times 10^3, & \text{if } \lambda_f < \mu_f \\ K, & \text{otherwise} \end{cases} \tag{11}$$

where K is a large constant that represents the delay of packets in the queue of forwarding node f when the system is unstable, meaning that the arrival rate of packets is higher than the service rate causing the queue to grow indefinitely. $\lambda_f = T_{children,f}$ and $\mu_f = T_f$ are the arrival rate and service rate

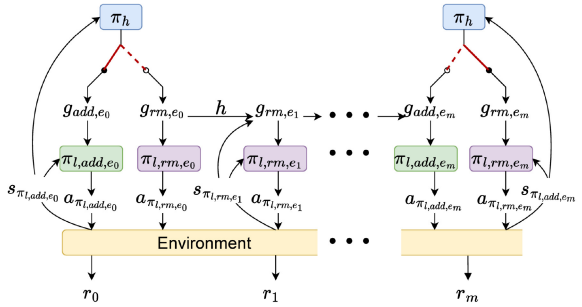

Fig. 2. Reinforcement learning architecture.

of packets at forwarding node f, respectively. Therefore, the delay of the network can be expressed as follows:

$$D = \frac{\sum_{n\in\mathcal{N}}\left(D_n + \xi_{d,n}\right)}{|\mathcal{N}|} \tag{12}$$

$$\text{subject to} \qquad |\mathcal{N}| > 0, \lambda_f/\mu_f < 1, K >> 0 \tag{13}$$

where $\xi_{d,n} \sim N(0, \sigma^2_{d,n})$ is a noise term that accounts for the uncertainty in the delay model of node n.

V. Reinforcement Learning

This section provides an overview of the HRL framework. It also discusses the reward function used to guide the learning process of the RL agents. Lastly, we discuss the action space and the RL algorithm used for training the agent.

A. Reinforcement Learning Overview

Designing an optimal TSCH schedule, which determines timeslots, channels, and slotframe size for efficient packet transmission and reception, is a challenging combinatorial optimization problem.

To tackle the complexity of the link assignments problem in a TSCH network, which intensifies with network size, timeslots and channels, and application requirements, we adopt a Hierarchical Reinforcement Learning (HRL) approach. This approach empowers us to solve the problem effectively through a trial-and-error learning process, where the agent discovers the optimal mapping between state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ that maximizes the cumulative reward $\mathcal{R}$.

B. Hierarchical Reinforcement Learning Architecture

We extend the RL framework to a HRL framework to solve the link assignments problem in a TSCH network with one higher-level policy π_h and multiple lower-level policies $\pi_{l,e}$ as shown in Fig. 2. Below we discuss the cost function used to guide the learning process of the RL agents.

1) Cost Function: The cost function is a function that maps the state $s \in \mathcal{S}$ and the action $a \in \mathcal{A}$ to a cost $c \in \mathcal{C}$, i.e., $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Let $\varphi = (\alpha, \beta, \gamma)$ be the tuple of application requirements, where α, β, and γ are the weights of the normalized power consumption $(\widehat{P})$, delay $(\widehat{D})$, and throughput $(\widehat{T})$ respectively. Therefore, the cost function can be expressed as follows:

$$c(s,a) = \alpha \times \widehat{P_t} + \beta \times \widehat{D_t} - \gamma \times \widehat{T_t} \tag{14}$$

$$\text{subject to} \quad \alpha + \beta + \gamma = 1 \tag{15}$$

The values of φ are provided by the application layer when the RL agent is doing the inference. During the learning process, the environment generates random φ to train the RL agent. Maximum performance is achieved when the cost c is minimized as shown below:

$$\min_{a\in\mathcal{A}} \quad c(s,a)$$
$$\text{subject to} \quad (5), (9), (13), (15) \tag{16}$$

We have now introduced the cost function used to guide the learning process of the RL agent. We will now focus on the two levels of the HRL architecture.

2) Higher-Level Policy: The higher-level policy π_h is responsible for selecting the optimal lower-level policy $\pi_{l,e}$, where $\pi_{l,e}$ is associated with a specific communication link $e \in \mathcal{E}$. In brief, π_h is in charge of selecting between two main actions:

1) *Add link:* This action adds a link e to the TSCH schedule.
2) *Remove link:* This action removes a link e from the TSCH schedule.

Thus, the action space $\mathcal{A}$ of the higher-level policy π_h is defined as follows:

$$\mathcal{A}_{\pi_h} = \left\{ a_{\pi_h} \mid a_{\pi_h} \in \left\{ \mathcal{A}_{\pi_h,add}, \mathcal{A}_{\pi_h,rm} \right\} \right\} \tag{17}$$

The terms $\mathcal{A}_{\pi_h,\text{add}}$ and $\mathcal{A}_{\pi_h,\text{rm}}$ represent specific subsets of the action space $\mathcal{A}_{\pi_h}$. Recall that $\mathcal{E}$ is the set of all possible communication links. We can define $\mathcal{A}_{\pi_h,\text{add}}$ as the set of elements $a_{\pi_h,\text{add}}$ such that each element represents a combination of communication links e chosen from $\mathcal{E}$. Formally, $\mathcal{A}_{\pi_h,\text{add}} = \{a_{\pi_h,\text{add}} \mid \forall e \in \mathcal{E}, a_{\pi_h,\text{add}}$ represents the combination of $e\}$. Thus, the size of the subset $\mathcal{A}_{\pi_h,add}$ is $|\mathcal{E}|$. Similarly, $\mathcal{A}_{\pi_h,\text{rm}}$ can be defined as the set of elements $a_{\pi_h,\text{rm}}$ such that each element represents a combination of communication links e chosen from $\mathcal{E}$ for removal. Formally, $\mathcal{A}_{\pi_h,\text{rm}} = \{a_{\pi_h,\text{rm}} \mid \forall e \in \mathcal{E}, a_{\pi_h,\text{rm}}$ represents the removal of $e\}$. The size of the subset $\mathcal{A}_{\pi_h,rm}$ is also $|\mathcal{E}|$. Therefore, the size of the action space $\mathcal{A}_{\pi_h}$ is $2 \times |\mathcal{E}|$. The state space $\mathcal{S}$ of the higher-level policy π_h is defined as follows:

$$\mathcal{S}_{\pi_h} = \left\{ s_{\pi_h} = \left(\widehat{P}, \widehat{D}, \widehat{T}, \varphi, \widehat{\mathcal{W}}, \widehat{\mathcal{H}}, \widehat{e} \right) \right\} \tag{18}$$

where $\widehat{\mathcal{W}}$ is a normalized adjacency array of $\mathcal{N}^2$ elements representing the network topology. $\widehat{\mathcal{H}}$ is a normalized array of $|\mathcal{Z}| \times |\mathcal{U}|$ elements representing the TSCH schedule. $\widehat{e}$ is the normalized value of the link e. s_{π_h} is provided by the environment to the RL agent at each time step t.

The reward function of policy π_h is defined as a function of the action taken, the state s_{π_h}, and the cost function c. To discourage the policy π_h from selecting actions that lead to higher costs or actions that are not feasible, we define a set of penalized actions $\mathcal{P}_{\pi_h}$. One type of penalized action for the policy π_h is denoted as $\mathcal{P}_{\pi_{h,add}} \subset \mathcal{P}_{\pi_h}$. This type of action corresponds to adding a link e that does not exist in the current forwarding paths $w \in \mathcal{W}$. Formally, $\mathcal{P}_{\pi_{h,add}} = \{p_{\pi_{h,add}} \mid p_{\pi_{h,add}} \in \mathcal{A}_{\pi_h,\text{add}}, \forall e \in p_{\pi_h}, \neg\exists w \in \mathcal{W}, e \in q\}$ The condition $p_{\pi_h} \in \mathcal{A}_{\pi_h,\text{add}}$ ensures that the penalized action

p_{π_h} is one of the actions in the set $\mathcal{A}_{\pi_h,\text{add}}$. The condition $\forall e \in p_{\pi_h}$ ensures that all the links e being added in the action are considered. Finally, the condition $\neg\exists w \in \mathcal{W}, e \in q$ ensures that there is no forwarding path q in the set Q that contains any of the links e being added in the penalized action.

Another type of penalized action for the policy π_h is denoted as $\mathcal{P}_{\pi_{h,rm}} \subset \mathcal{P}_{\pi_h}$. This type of action corresponds to removing a link e such as removing the link e would result in a forwarding path $w \in \mathcal{W}$ that does not contain any links. Formally, $\mathcal{P}_{\pi_{h,rm}} = \{p_{\pi_{h,rm}} \mid p_{\pi_{h,rm}} \in \mathcal{A}_{\pi_h,\text{rm}}, \forall e \in p_{\pi_h}, \exists w \in \mathcal{W}, e \in q\}$. The condition $\exists w \in \mathcal{W}, e \in q$ ensures that there is a forwarding path q in the set Q that contains any of the links e being removed in the penalized action. We can now define the set of penalized actions as $\mathcal{P}_{\pi_h} = \{p_{\pi_h} \mid p_{\pi_h} \in \mathcal{P}_{\pi_{h,add}} \cup \mathcal{P}_{\pi_{h,rm}}\}$.

The policy π_h receives a penalty ψ_{π_h} when it selects a penalized action $p_{\pi_h} \in \mathcal{P}_{\pi_h}$, and the episode ends as the agent has reached a terminal state. The agent also reaches a terminal state when it has reached the maximum number of steps T_{π_h}. Consequently, the immediate reward r of the policy π_h is defined as follows:

$$r_{\pi_h}(s_{\pi_h}, a_{\pi_h}) = \begin{cases} \psi_{\pi_h}, & \text{if } a_{\pi_h} \in \mathcal{P}_{\pi_h} \\ \upsilon - c(s_{\pi_h}, a_{\pi_h}), & \text{otherwise} \end{cases} \quad (19)$$

where $\upsilon > c(s_{\pi_h}, a_{\pi_h})$. The objective is to find a policy $\pi_h^* = \arg\max_{\pi_h} \mathcal{R} = \arg\max_{\pi_h} \sum_{t=0}^{T} \lambda^t r_{\pi_h}(s_{\pi_h}, a_{\pi_h})$ that maximizes the cumulative reward $\mathcal{R}$ over a time horizon. Where λ is the discount factor that determines the importance of future rewards.

We have now introduced the higher-level policy π_h, its action space, state space, and reward function. We will now discuss the lower-level policy $\pi_{l,e}$.

3) Lower-Level Policies: As discussed earlier, the action space of the policy π_h consists of $2 \times |\mathcal{E}|$ actions. To select the optimal cell $h = (u, \zeta)$, where $u \in \mathcal{U}$ and $\zeta \in \mathcal{Z}$, for each action a_{π_h}, we need to define a set of lower-level policies. Let $\mathcal{L}$ denote the set of lower-level policies.

The set $\mathcal{L}$ consists of $2 \times |\mathcal{E}|$ lower-level policies, denoted as $\pi_{l,e}$, where $e \in \mathcal{E}$. Among these lower-level policies, $|\mathcal{E}|$ are dedicated to selecting the optimal slot h for adding link e to the TSCH schedule, while the remaining $|\mathcal{E}|$ policies are responsible for selecting the optimal slot h to remove link e from the TSCH schedule. By defining the set $\mathcal{L}$, we can effectively handle the selection of optimal cells at the lower level within the hierarchical architecture.

Both types of lower-level policies share the same action space $\mathcal{A}_{\pi_{l,e}}$ and state space $\mathcal{S}_{\pi_{l,e}}$ for their respective link e. The action space $\mathcal{A}$ of the lower-level policies $\pi_{l,e}$ is defined as $\mathcal{A}_{\pi_{l,e}} = \{a_{\pi_{l,e}} \mid a_{\pi_{l,e}} \in \mathcal{H}\}$, where $\mathcal{H}$ is the set of all possible cells $h = (u, \zeta)$. It is important to note that the action space $\mathcal{A}_{\pi l,e}$ is specific to each link e. The number of actions in $\mathcal{A}_{\pi l,e}$ is equal to the number of cells in the TSCH schedule, which is given by $|\mathcal{Z}| \times |\mathcal{U}|$ for each link e.

The state space $\mathcal{S}$ of the lower-level policy $\pi_{l,e}$ for each link e is defined as follows:

$$\mathcal{S}_{\pi_{l,e}} = \left\{ s_{\pi_{l,e}} = \left(\widehat{P}, \widehat{D}, \widehat{L}, \varphi, \widehat{\mathcal{W}}, \widehat{\mathcal{H}} \times k, \widehat{e} \right) \right\} \quad (20)$$

Here, the state $s_{\pi_{l,e}}$ is provided by the environment to the RL agent at each time step t. $\widehat{\mathcal{H}} \times k$ is an array that represents the TSCH schedule of the source and destination nodes and the network schedule occupation. It is important to note that the state space $\mathcal{S}_{\pi_{l,e}}$ is specific to each link e.

The reward function of the two types of lower-level policies $\pi_{l,e}$ differs in the penalized set of actions $\mathcal{P}_{\pi_{l,e}}$. We denote the set of occupied cells in state $s_{\pi_{l,e}}$ as $H(s_{\pi_{l,e}})$. For the policy $\pi_{l,add}$, we define a set of penalized actions $\mathcal{P}_{l,add} = \{a_{\pi_{l,e}} \mid a_{\pi_{l,e}} = h = (u, \zeta), u \in \mathcal{U}, \zeta \in \mathcal{Z}, h \in H(s_{\pi_{l,e}})\}$ to discourage the selection of infeasible or suboptimal actions. The penalized actions include selecting cells that are already occupied in the current TSCH schedule, adding a transmission link to a cell already occupied by the source node, or adding a reception link to a cell already occupied by the destination node. By including the penalized set $\mathcal{P}_{l,add}$ in the learning, we guide the agent towards policies that prioritize optimal cell selection while avoiding actions that may have negative consequences on system performance. Besides, it promotes the exploration of more efficient and non-overlapping solutions.

For the policy $\pi_{l,rm}$, we define a set of penalized actions $\mathcal{P}_{l,\text{rm}} = \{a_{\pi_{l,e}} \mid a_{\pi_{l,e}} = h = (u, \zeta), u \in \mathcal{U}, \zeta \in \mathcal{Z}, h \notin H(s_{\pi_{l,e}}) \text{ or } m(h) \neq m(s_{\pi_{l,e}}(h))\}$. Where $m(h)$ represents the destination node associated with slot h, and $m(s_{\pi_{l,e}}(h))$ represents the destination node associated with the scheduled link in cell h. The condition $h \notin H(s_{\pi_{l,e}})$ ensures that the action selects a cell that is already occupied, preventing the agent from attempting to remove a nonexistent link from an empty cell. The condition $m(h) \neq m(s_{\pi_{l,e}}(h))$ checks if the link in cell h points to a different destination node. If this condition is met, it indicates a mismatch between the scheduled link configuration and the agent's removal action, which could disrupt the established communication paths and lead to suboptimal performance. By penalizing actions in $\mathcal{P}_{l,\text{rm}}$, we guide the agent to focus on removing redundant links only from valid cells while avoiding actions that could cause disruptions or inconsistencies in the network topology. We can now define the set of penalized actions in the lower-level policy $\pi_{l,e}$ as $\mathcal{P}_{\pi_{l,e}} = \{p_{\pi_{l,e}} \mid p_{\pi_{l,e}} \in \mathcal{P}_{l,add} \cup \mathcal{P}_{l,\text{rm}}\}$.

The immediate reward r of the lower-level policy $\pi_{l,e}$ is defined as follows:

$$r_{\pi_{l,e}}\left(s_{\pi_{l,e}}, a_{\pi_{l,e}}\right) = \begin{cases} s\psi_{\pi_l}, & \text{if } a_{\pi_{l,e}} \in \mathcal{P}_{\pi_{l,e}} \\ \upsilon - c\left(s_{\pi_{l,e}}, a_{\pi_{l,e}}\right), & \text{otherwise} \end{cases} \quad (21)$$

The objective here is also to find a policy $\pi_{l,e}$ that maximizes the cumulative reward $\mathcal{R}$ over a time horizon, as in π_h.

C. Training

The HRL-TSCH algorithm undergoes training using the DQN algorithm, as illustrated in Algorithm 1. It is important to note that the lower-level policies $\pi_{l,e}$ are trained independently of each other. Once the lower-level policies $\pi_{l,e}$ are successfully trained, the higher-level policy π_h is then trained.

The training process for the lower-level policies $\pi_{l,e}$ closely mirrors the training process for the higher-level policy π_h.

Algorithm 1: HRLTSCH Algorithm

Input: Replay memory capacity $\mathcal{D}$, batch size $\mathcal{B}$, target network update rate α_{θ^-}, discount factor λ, initial exploration rate ϵ, minimum exploration rate $\epsilon_{\min}$, and exploration rate decay ϵ_{decay}
Notations: θ and θ^- are the weights of the Q-network and target network, respectively.
Initialize replay memory $\mathcal{D}$, parameters θ and θ^- with random weights;
for *Each episode* **do**
  Reset the environment $(\mathcal{H}, \varphi)$;
  Observe the state s_{π_h} from the environment;
  while *not done* **do**
    $\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon_{\text{decay}} \times \epsilon)$;
    Choose action a_{π_h} using ϵ-greedy policy;
    /* Select the optimal link e to be added or removed */
    if $a_{\pi_h} \in \mathcal{A}_{\pi_h,\text{add}}$ **then**
      /* Select the optimal cell for link e to be added */
      Choose action $a_{\pi_{l,e}}$ using ϵ-greedy policy;
    end
    if $a_{\pi_h} \in \mathcal{A}_{\pi_h,\text{rm}}$ **then**
      /* Select the optimal link e to be removed */
      Choose action $a_{\pi_{l,e}}$ using ϵ-greedy policy;
    end
    Execute action $a_{\pi_{l,e}}$, obtain reward $r_{\pi_{l,e}}$ using Eq. (21) and next state $s'_{\pi_{l,e}}$;
    Obtain reward r_{π_h} using Eq. (19);
    Store transition $(s_{\pi_h}, a_{\pi_h}, r_{\pi_h}, s'_{\pi_h})$ in $\mathcal{D}$;
    if $|\mathcal{D}| > |\mathcal{B}|$ **then**
      Sample a batch (s, a, r, s') from $\mathcal{D}$;
      /* Calculate the loss and update the weights of the Q-network */
      $L(\theta) \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left(y_{\pi_h} - Q(s_{\pi_h}, a_{\pi_h}; \theta)\right)^2$;
      Update the weights of the Q-network;
      if $\text{mod}(t, \alpha_{\theta^-}) = 0$ **then**
        Update the weights of the target network $\theta^- \leftarrow \theta$;
      end
    end
  end
end

Algorithm 2: TSCH Link Selection Algorithm

Function `get_ts_ch_from_dst_addr(dst)`
  Input : Destination address *dst*
  Output: u, ζ Initialize *diff*, *min* $\leftarrow \infty$, $|\mathcal{U}|$;
  /* Calculate the current time slot u */
  $u_{ASN} \leftarrow ASN \bmod |\mathcal{U}|$;
  /* Iterate through the list of links e */
  $l \leftarrow$ head of links e list;
  while $l \neq \emptyset$ **do**
    /* Check if the destination address of the link e matches the provided destination address */
    if $dst = l.dst$ **then**
      /* Get the time slot u with the minimum difference */
      diff $\leftarrow l.u - u_{ASN}$;
      if *diff*< 0 **then**
        /* We add to the difference the number of time slots $|\mathcal{U}|$ as the time slot u is in the past */
        diff $+= |\mathcal{U}|$;
      end
      if *diff* $<$ *min* **then**
        /* Update the minimum difference, store the time slot u and channel offset ζ */
        min $\leftarrow$ *diff* ;
        $u \leftarrow l.u$;
        $\zeta \leftarrow l.\zeta$;
      end
    end
  end
  $l \leftarrow$ next link e in the list;
  /* Return the time slot u and channel offset ζ */
  return u, ζ;
end

Following the training of the lower-level policies, the higher-level policy π_h utilizes these trained policies to make informed decisions, selecting optimal actions $a_{\pi_{l,e}}$ for each link e.

We trained the RL agents with 5×10^5 steps using a replay memory capacity of 10^5, a batch size of 512, a learning rate (σ) of 0.001, a learning start time of 5000, a discount factor (λ) of 0.8, an exploration fraction of 0.7, and a minimum exploration rate of 0.01.

D. TSCH Lookup Algorithm

Once the RL agents select the optimal links e and cells h, the next step is to generate the TSCH schedule and distribute it to all $n \in \mathcal{N}$. The TSCH schedule is a list of e, where each e is associated with a source node, a destination node, a timeslot, and a channel offset. The SDN controller is responsible for translating the TSCH schedule into packets that are sent to the nodes in the network. Each node $n \in N$ in the network receives the TSCH schedule and processes it to generate the TSCH schedule for its use.

For the low-level policy, Each $n \in N$ runs the Algorithm 2 to determine the u and ζ to use. This algorithm serves as a complementary mechanism to the RL-driven decision-making process. While RL agents determine the optimal links and cells to use, the TSCH lookup algorithm ensures that the selected links are correctly scheduled in the TSCH network. It performs an iterative search through the list of e, comparing the destination address of each link with the provided destination address. The algorithm calculates the difference between the u of each e and the current Absolute Slot Number (ASN), and it keeps track of the minimum difference found and stores the corresponding u and ζ.

VI. Performance Evaluation

We conducted our experiments in the Cooja simulator [33] with retransmissions disabled to approximate the network closely to our model. The simulations run on a single machine with an Intel Core i9 CPU with 16GB of RAM. The network topology represents a small-scale network comprising ten SNs, as illustrated in Fig. 3(a). Each SN is 30 meters apart from its neighbors, and the sink node is located at the top of the network. All sensor nodes are equipped with an IEEE 802.15.4 radio transceiver and use the Unit Disk Graph Medium (UDGM) as the distance loss model. The transmission and interference range of the radio transceiver is set to 50 and 100 meters, respectively (see Fig. 3(b)). SNs are equipped with the Contiki-NG Energest module to calculate the energy consumption of SNs at each power state (e.g., transmit, receive, listen, and sleep). In this configuration,

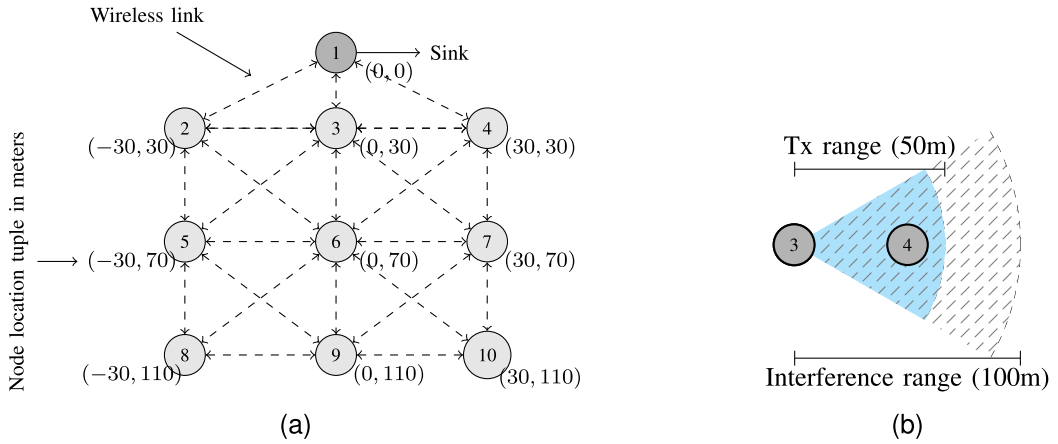

Fig. 3. Network topology and wireless link visualization. (a) Network topology with the sink node and sensor nodes. (b) Wireless link visualization showing the transmission range and interference range.

TABLE III
NETWORK PARAMETERS

Parameter	Value	Parameter	Value
$\|\mathcal{N}\|$	10	E_{tx_ack}	55 μJ
$\|\mathcal{U}\|$	17	E_{rx}	160 μJ
$\|\mathcal{Z}\|$	2	E_{rx_ack}	70 μJ
$\|u\|$	10 ms	E_{listen}	110 μJ
ψ_{π_h}, ψ_{π_l}	-1	Operating voltage	3 V
υ	2	Tx current	19.5 mA
K	10^3 ms	Rx current	21.8 mA
E_{tx}	140 μJ	CPU current	1.85 mA
Deep LPM current	0.0051 mA	LPM current	0.0545 mA
Dist. between SNs	30 m	Propagation model	UDGM
Tx range	50 m	If range	100 m
Data interval	1 s	Packet size	12 B

TABLE IV
PROTOCOL PARAMETERS

Protocol	SF	φ	ReTx	Notation
HRL-TSCH	17	(0.5, 0.3, 0.2)	✗	HRL-TSCH(φ_1)
HRL-TSCH	17	(0.1, 0.5, 0.4)	✗	HRL-TSCH(φ_2)
HRL-TSCH	17	(0.2, 0.2, 0.6)	✗	HRL-TSCH(φ_3)
HRL-TSCH	17	(0.8, 0.1, 0.1)	✗	HRL-TSCH(φ_4)
HRL-TSCH	17	(0.1, 0.8, 0.1)	✗	HRL-TSCH(φ_5)
HRL-TSCH	17	(0.1, 0.1, 0.8)	✗	HRL-TSCH(φ_6)
HRL-TSCH	17	(0.5, 0.3, 0.2)	✓	HRL-TSCH(φ_1^{ReTx})
HRL-TSCH	17	(0.1, 0.5, 0.4)	✓	HRL-TSCH(φ_2^{ReTx})
HRL-TSCH	17	(0.2, 0.2, 0.6)	✓	HRL-TSCH(φ_3^{ReTx})
HRL-TSCH	17	(0.8, 0.1, 0.1)	✓	HRL-TSCH(φ_4^{ReTx})
HRL-TSCH	17	(0.1, 0.8, 0.1)	✓	HRL-TSCH(φ_5^{ReTx})
HRL-TSCH	17	(0.1, 0.1, 0.8)	✓	HRL-TSCH(φ_6^{ReTx})
Orchestra	11	-	✗	Orchestra-11
Orchestra	11	-	✓	Orchestra-11-ReTx
MSF	3	-	✗	MSF-3
MSF	3	-	✓	MSF-3-ReTx
MSF	5	-	✗	MSF-5
MSF	5	-	✓	MSF-5-ReTx
QL-TSCH-3	3	-	✗	QL-TSCH-3
QL-TSCH-3	3	-	✓	QL-TSCH-3-ReTx
QL-TSCH-5	5	-	✗	QL-TSCH-5
QL-TSCH-5	5	-	✓	QL-TSCH-5-ReTx

the sink node is connected to the control plane via a serial interface, as depicted in Fig. 1. We utilize this small-scale network to avoid excessive complexity in our experiments and to enable us to draw meaningful conclusions from the proof of concept of the proposed approach. A summary of the network parameters is provided in Table III.

A. Baselines

We compare the performance of our approach with the following baselines:

1) We use the Orchestra [24] implementation in the Contiki-NG operating system [34] as a baseline. Orchestra autonomously schedules the TSCH links with little overhead and high reliability.
2) We utilize the Minimal Scheduling Function (MSF), an established Internet Engineering Task Force (IETF) standard for TSCH scheduling, as a baseline. The MSF orchestrates the TSCH links within a unified shared cell, optimizing for minimal connectivity [35].
3) We also compare our approach with the QL-based TSCH scheduler [23] as a baseline.

The configurations and notations of the protocols are summarized in Table IV. This table provides details such as the Slotframe Size (SF) utilized in the TSCH schedule, the optimized φ combination for enhancing network performance, and whether retransmissions are enabled. The selection of SF for MSF and QL-TSCH is based on the required number of slots to accommodate network traffic. The default number of retransmissions is set to 3.

B. Pareto Front

The Pareto front is constructed by systematically varying the user requirements, employing a finely tuned step size of 0.1 for precision. We conducted 66 distinct simulations, each with a unique φ combination. Each simulation lasted 40 minutes, yielding over 16k packets. Refer to Fig. 4 for a visualization of the power consumption, delay, and throughput trade-offs. To benchmark against the baselines, our strategy focuses on prioritizing φ combinations that optimize trade-offs between power consumption and delay, throughput and delay, and power consumption and throughput. The φ values that achieve the most favorable balance in these trade-offs

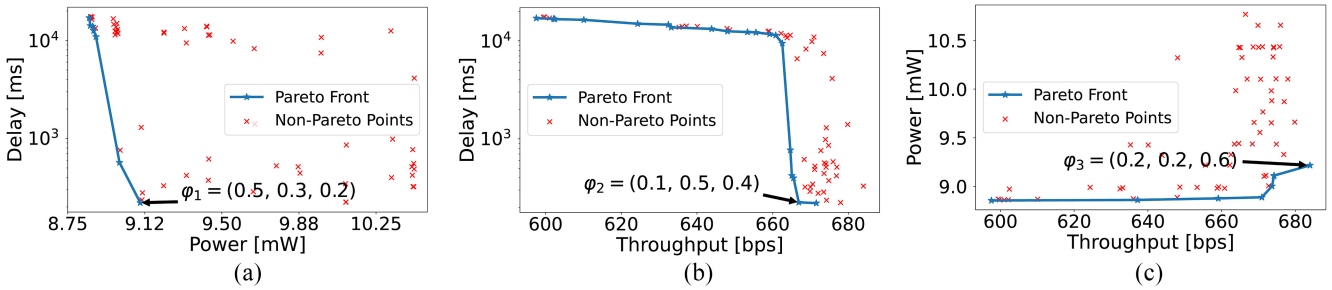

Fig. 4. Pareto front that shows the trade-offs between power consumption, delay, and throughput. Fig. 4(a) shows the trade-offs between power consumption and delay. Fig. 4(b) shows the trade-offs between throughput and delay. Fig. 4(c) shows the trade-offs between throughput and power consumption.

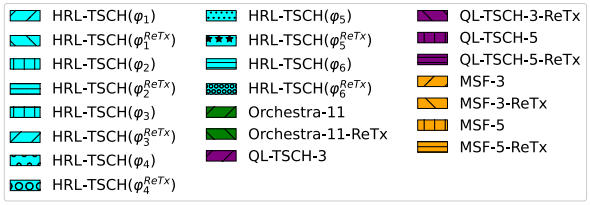

Fig. 5. Common legend for Fig. 6, Fig. 7, Fig. 8, Fig. 10, Fig. 11 and Fig. 12.

are $\varphi_1 = (0.5, 0.3, 0.2)$, $\varphi_2 = (0.1, 0.5, 0.4)$, and $\varphi_3 = (0.2, 0.2, 0.6)$, respectively. Moreover, we explore scenarios where one objective is emphasized over the others, represented by $\varphi_4 = (0.8, 0.1, 0.1)$, $\varphi_5 = (0.1, 0.8, 0.1)$, and $\varphi_6 = (0.1, 0.1, 0.8)$.

C. Comparison With Baselines

1) Network Power Consumption: Fig. 6(a) illustrates the network's average power consumption. Among all protocols, MSF and QL-TSCH exhibit the highest power consumption, approximately two to three times higher than HRL-TSCH. MSF requires sensor nodes to remain in an active state throughout the network operation, while QL-TSCH's action peeking algorithm schedules receiving slots in all slots except the transmitting slot. Notably, HRL-TSCH achieves lower power consumption for the φ_4 combination compared to the other φ combinations, as it prioritizes power consumption over delay and throughput.

Given that HRL-TSCH allocates some attention to delay and throughput across all φ combinations, it tends to consume more power than Orchestra, a correlation directly influenced by the requirements of delay and throughput. However, it's crucial to underscore that the substantial performance enhancements realized with HRL-TSCH justify its slightly higher power consumption. Orchestra, conversely, optimizes consumption through receiver-based scheduling, employing fewer slots compared to HRL-TSCH. Despite this efficiency, Orchestra's approach significantly increases its Packet Loss Rate (PLR) to approximately 65% (almost five times higher than HRL-TSCH), contrasting with HRL-TSCH's lower PLR of approximately 12% for the worst φ combination, as depicted in Fig. 7. Furthermore, it's noteworthy that HRL-TSCH operates in a contention-free manner, unlike Orchestra.

2) Power Consumption Per Node: Fig. 8 illustrates the power consumption per node for HRL-TSCH and the baseline protocols. Among the baselines, MSF and QL-TSCH demonstrate the highest power consumption per node. Specifically, QL-TSCH consumes approximately twice as much power per node compared to HRL-TSCH. Similarly, MSF exhibits roughly twice the power consumption per node compared to HRL-TSCH, except for nodes eight, nine, and ten, where MSF consumes approximately four times as much power. In contrast, HRL-TSCH demonstrates a more balanced power consumption per node across all nodes, with only some deviations observed at nodes three and six. These nodes are tasked with forwarding packets from deeper nodes within the network.

3) End-to-End Latency: Fig. 6(b) illustrates the average delay experienced by the network, revealing distinct performance characteristics. Notably, protocols with the lowest slotframe size, such as MSF-3 and QL-TSCH-3, demonstrate the lowest delay among all protocols. This outcome is unsurprising, given that a lower slotframe size entails fewer slots to traverse, thereby reducing delay. However, this efficiency comes at the expense of higher power consumption, as indicated in Fig. 6(a). Additionally, it's important to highlight that while MSF and QL-TSCH achieve the lowest delay when a packet successfully traverses all hops in the network, this occurrence is infrequent, as shown in Fig. 7.

Specifically, HRL-TSCH outperforms Orchestra by achieving a consistently lower delay. This superiority can be attributed to HRL-TSCH's capacity to dynamically adapt the TSCH schedule in response to changing network conditions—an adaptability that Orchestra lacks. This adaptability ensures that HRL-TSCH remains effective in minimizing delays across varying network states, contributing to its enhanced performance in comparison to Orchestra.

4) Analysis of Per-Node End-to-End Latency for HRL-TSCH: Fig. 9 illustrates the end-to-end delay per node for each φ combination. All charts share the same scale to facilitate comparison. The dashed green line represents the average delay of the network. The φ_5 combination achieves the lowest delay for all nodes, as it prioritizes delay over power consumption and throughput. The φ_3 combination achieves the second-lowest delay for all nodes, as it prioritizes throughput over power consumption and delay, thus scheduling more slots for data transmission. The highest delay and deviation are observed for nodes eight, nine, and ten, located at the

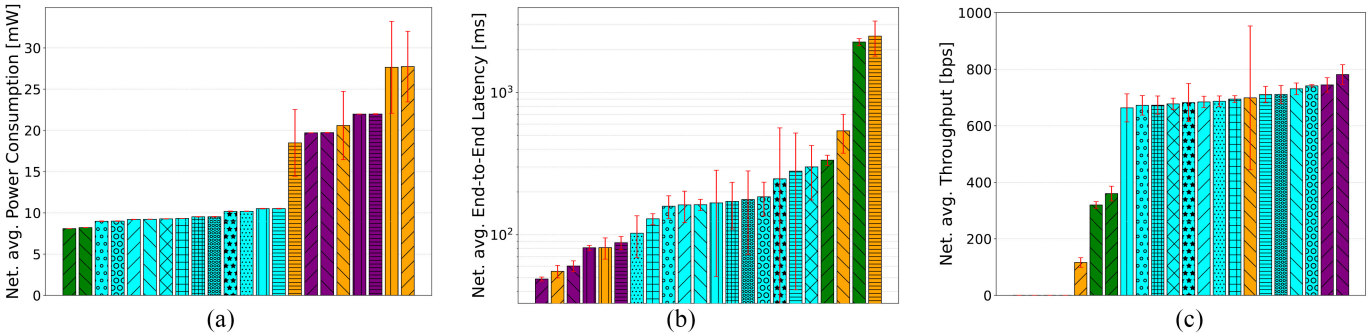

Fig. 6. Performance comparison of HRL-TSCH and baselines for network power consumption, delay, and throughput (see Fig. 5 for the legend). (a) shows the average power consumption of the network. (b) shows the average delay of the network. (c) shows the average throughput of the network.

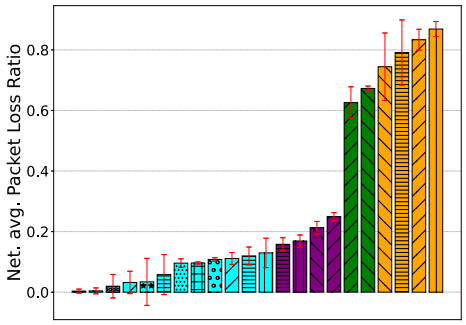

Fig. 7. PLR of HRL-TSCH and baselines (see Fig. 5 for the legend).

edge of the network, requiring more hops to reach the sink node.

5) Throughput: The average throughput of the network, depicted in Fig. 6(c), highlights distinct performance advantages among the scheduling approaches. Notably, HRL-TSCH emerges as the top performer, consistently achieving higher throughput compared to Orchestra, MSF, and QL-TSCH.

Orchestra lags behind in throughput performance (approximately twice lower than HRL-TSCH), primarily attributed to its receiver-based scheduling approach. This method imposes limitations on the number of receiving slots within the TSCH schedule, constraining the overall throughput potential. Furthermore, MSF-3 and MSF-5 demonstrate poor throughput performance. The shared cell approach in MSF limits concurrent transmissions, resulting in lower throughput compared to HRL-TSCH. Conversely, QL-TSCH-3 and QL-TSCH-3-ReTx achieve the highest throughput among all MSF and QL-TSCH configurations. This is attributed to QL-TSCH's scheduling of receiving slots in all slots except the transmitting slot, facilitating increased concurrent transmissions. However, QL-TSCH-5 and QL-TSCH-5-ReTx exhibit the lowest throughput due to an increased slotframe size, which fails to accommodate the network traffic. Consequently, HRL-TSCH stands out as the superior choice for achieving higher and more adaptable throughput in dynamic IIoT environments.

6) Analysis of Per-Node Throughput for HRL-TSCH: Fig. 10 illustrates the throughput per node for each φ combination as well as the baseline protocols. For all nodes except nodes three and six, all φ combinations achieve similar throughput, slightly lower than that of QL-TSCH-3. Notably, HRL-TSCH consistently achieves the highest throughput for nodes three and six across all φ combinations. This result aligns with expectations, as HRL-TSCH schedules more slots for data transmission compared to QL-TSCH-3, and nodes three and six typically handle more data traffic. Conversely, QL-TSCH-5 and MSF-5 consistently exhibit the lowest throughput for all nodes, indicating their inability to effectively manage network traffic demands.

7) Analysis of Per-Node Jitter: Fig. 11 presents the jitter analysis for each node. Jitter is computed as the time difference between the arrival of two consecutive packets. HRL-TSCH consistently outperforms other protocols across all φ combinations in terms of jitter. Its meticulous scheduling approach ensures minimal delay in packet transmission and reception, resulting in consistently low jitter. Conversely, MSF-3 and MSF-5 exhibit the highest jitter among all protocols. This is primarily attributed to the shared cell approach in MSF, which restricts access to the shared cell, consequently leading to heightened jitter.

8) Analysis of Per-Node PLR: The PLR analysis for each node is depicted in Fig. 12. Fig. 12(a) displays the PLR without retransmissions, while Fig. 12(b) showcases the PLR with retransmissions enabled. The PLR is calculated as the ratio of lost packets to the total number of transmitted packets. HRL-TSCH outperforms all baseline protocols in terms of PLR across all nodes. Particularly noteworthy is its superior performance at edge nodes, where the PLR tends to be higher due to the increased number of hops. Notably, the PLR is significantly higher for Orchestra, MSF, and QL-TSCH compared to HRL-TSCH, underlining the latter's effectiveness in minimizing packet loss. Furthermore, HRL-TSCH achieves the lowest PLR even without retransmissions enabled, further underscoring its robustness in maintaining packet integrity.

9) Protocol Ranking: In Fig. 13, a comprehensive representation of trade-offs between power consumption, delay, and throughput is depicted. The proximity of a protocol to the minimum values of power consumption and delay, while maximizing throughput, serves as a crucial indicator of its overall performance in balancing these trade-offs.

In this context, Fig. 13(a) illustrates the trade-offs between power consumption and delay. Protocols positioned closer to the origin of the power consumption and delay axes are deemed optimal. Additionally, protocols depicted with

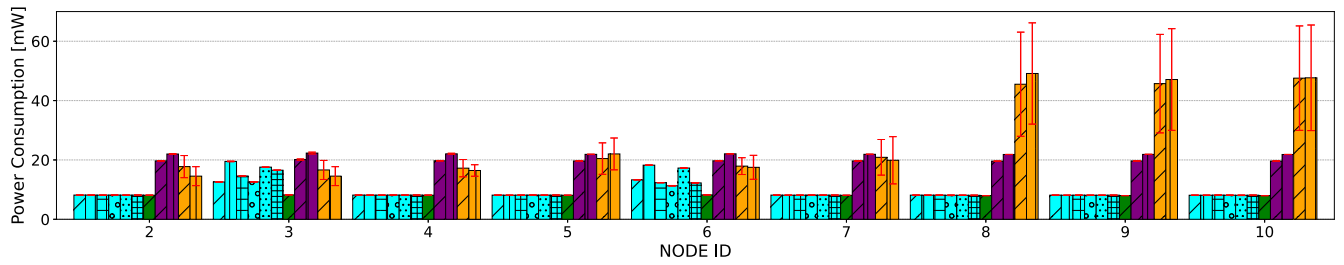

Fig. 8. Power consumption of HRL-TSCH and baselines for each node with retransmissions disabled (see Fig. 5 for the legend).

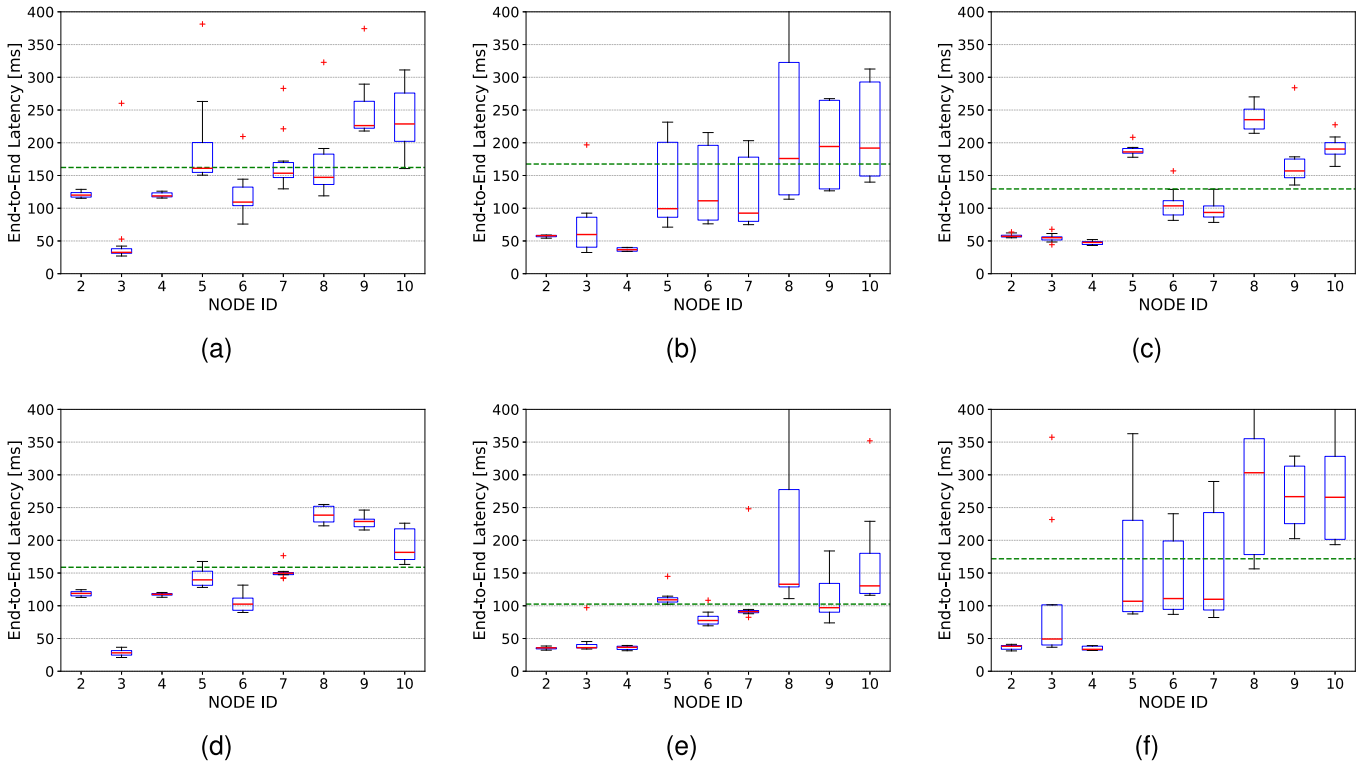

Fig. 9. End-to-end latency of HRL-TSCH for each node with retransmission disabled. (a) $\varphi_1 = (0.5, 0.3, 0.2)$. (b) $\varphi_2 = (0.1, 0.5, 0.4)$. (c) $\varphi_3 = (0.2, 0.2, 0.6)$. (d) $\varphi_4 = (0.8, 0.1, 0.1)$. (e) $\varphi_5 = (0.1, 0.8, 0.1)$. (f) $\varphi_6 = (0.1, 0.1, 0.8)$.

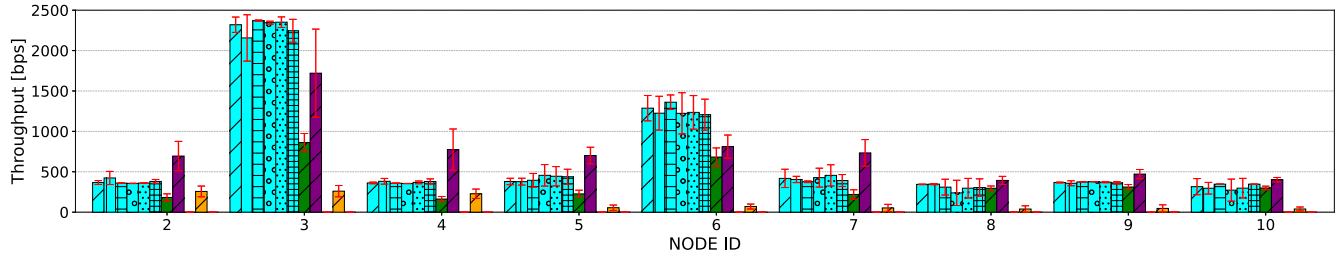

Fig. 10. Throughput of HRL-TSCH and baselines for each node with retransmissions disabled (see Fig. 5 for the legend).

a lighter shade of vivid color indicate higher throughput. It's evident that HRL-TSCH, across all φ combinations, is positioned closer to the origin compared to the baseline protocols. Furthermore, they consistently exhibit higher throughput compared to the baselines.

Fig. 13(b) visualizes the trade-offs between power consumption and throughput. In this representation, protocols closer to the y-axis signify lower power consumption, while those positioned further from the x-axis indicate higher throughput. Notably, HRL-TSCH configurations (clustered in

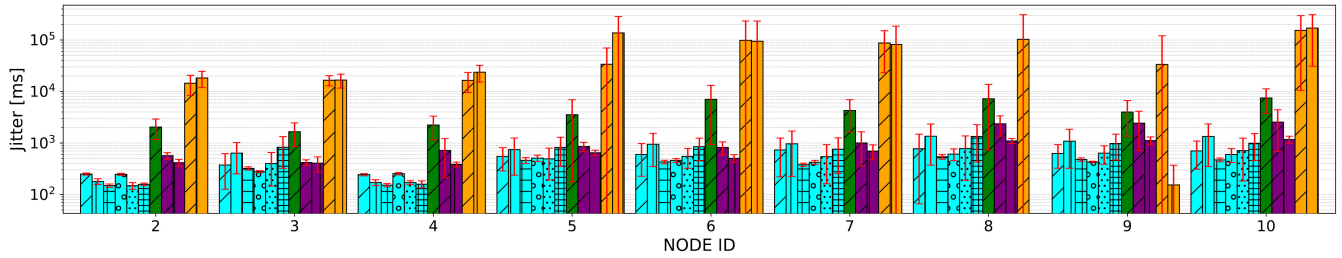

Fig. 11. Jitter of HRL-TSCH and baselines for each node with retransmissions disabled (see Fig. 5 for the legend).

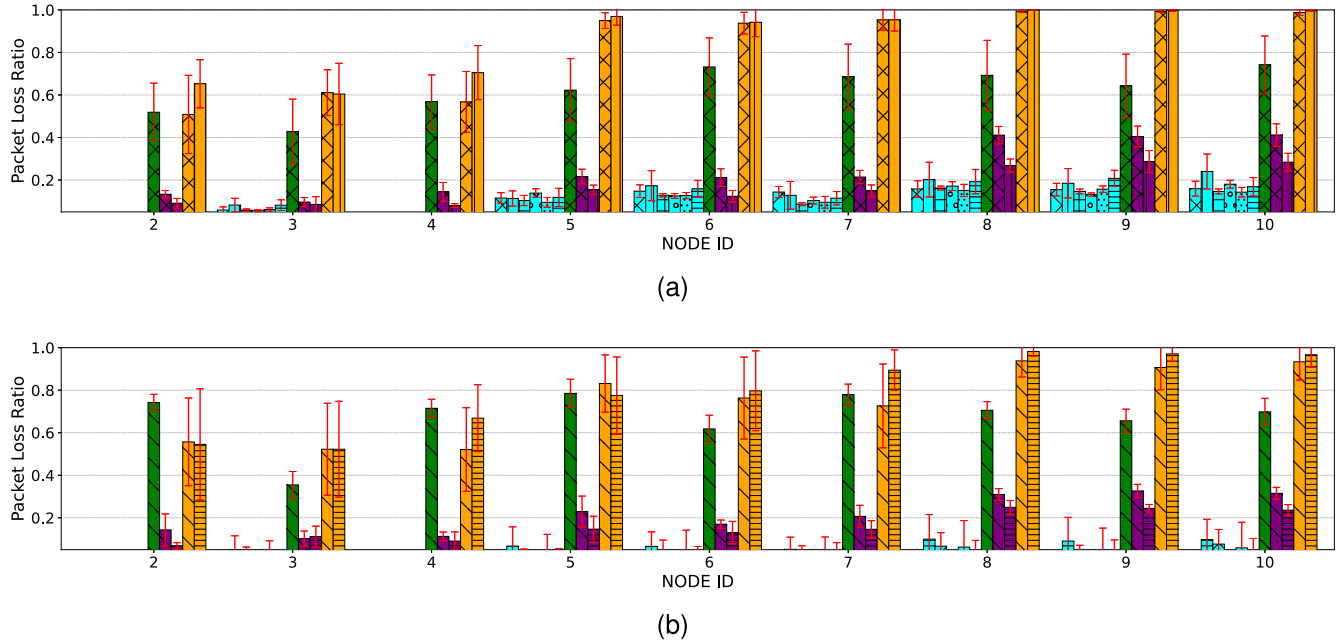

Fig. 12. PLR of HRL-TSCH and baselines for each node (see Fig. 5 for the legend). (a) PLR without retransmissions. (b) PLR with retransmissions.

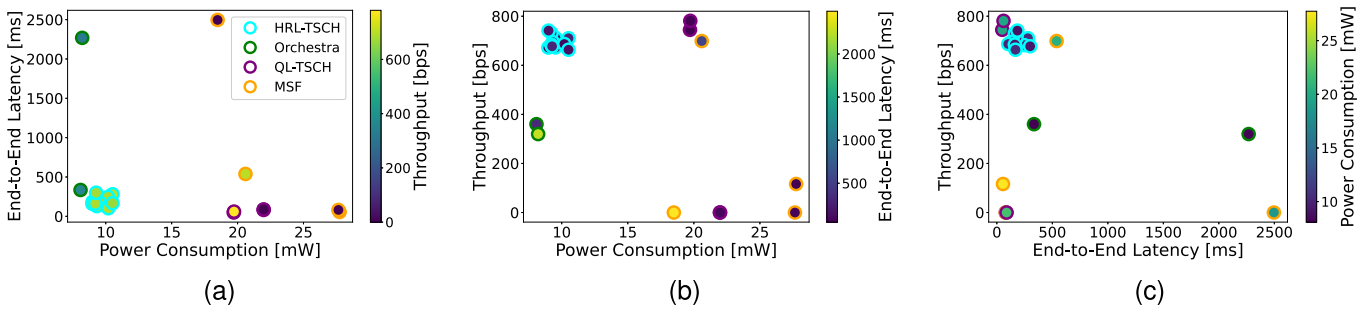

Fig. 13. Heatmap that shows the trade-offs between power consumption, delay, and throughput. (a) shows the trade-offs between power consumption and delay. (b) shows the trade-offs between power consumption and throughput. (c) shows the trade-offs between throughput and delay.

the top-left quadrant) are situated closer to the y-axis and further from the x-axis compared to the baseline protocols. This positioning underscores that HRL-TSCH achieves superior throughput with lower power consumption relative to the baselines. Furthermore, across all φ combinations, HRL-TSCH demonstrates lower delay, consolidating its performance advantages.

Fig. 13(c) illustrates the trade-offs between delay and throughput. In this depiction, protocols positioned closer to the y-axis signify lower delay, while those situated further from the x-axis indicate higher throughput. Notably, all HRL-TSCH φ combinations are clustered in the top-left corner of the graph, indicating that HRL-TSCH achieves superior throughput with lower delay compared to the baseline protocols. Furthermore, across all φ combinations, HRL-TSCH exhibits lower power consumption, further solidifying its performance advantages.

To comprehensively evaluate the performance of each protocol, we adopt a multi-dimensional ranking approach, considering factors such as balanced performance, power efficiency, sensitivity to delay, throughput, and reliability within the network. Firstly, we assign scores to each protocol based on their performance across these metrics. We utilize linear

TABLE V
RANKING OF PROTOCOLS FOR A BALANCED, POWER-EFFICIENT, DELAY-SENSITIVE, THROUGHPUT-SENSITIVE, AND RELIABILITY-SENSITIVE NETWORK

Balanced (points)	Protocol	Power (points)	Protocol	Delay (points)	Protocol	Throughput (points)	Protocol	Reliability (points)	Protocol
96.17	HRL-TSCH(φ_4^{ReTx})	95.68	HRL-TSCH(φ_4^{ReTx})	95.51	HRL-TSCH(φ_1^{ReTx})	98.47	HRL-TSCH(φ_4^{ReTx})	95.4	HRL-TSCH(φ_4^{ReTx})
95.76	HRL-TSCH(φ_1^{ReTx})	94.84	HRL-TSCH(φ_1^{ReTx})	95.13	HRL-TSCH(φ_4^{ReTx})	98.26	HRL-TSCH(φ_1^{ReTx})	94.42	HRL-TSCH(φ_1^{ReTx})
94.14	HRL-TSCH(φ_6^{ReTx})	93.75	HRL-TSCH(φ_4)	95.12	HRL-TSCH(φ_5)	96.54	HRL-TSCH(φ_6^{ReTx})	92.26	HRL-TSCH(φ_6^{ReTx})
92.15	HRL-TSCH(φ_3)	93.25	HRL-TSCH(φ_6^{ReTx})	94.88	HRL-TSCH(φ_3)	94.74	HRL-TSCH(φ_3^{ReTx})	91.61	QL-TSCH-3-ReTx
91.78	HRL-TSCH(φ_3^{ReTx})	93.1	HRL-TSCH(φ_3^{ReTx})	94.51	HRL-TSCH(φ_6^{ReTx})	94.4	HRL-TSCH(φ_5^{ReTx})	90.84	HRL-TSCH(φ_2^{ReTx})
91.26	HRL-TSCH(φ_5^{ReTx})	93.08	HRL-TSCH(φ_3)	93.8	HRL-TSCH(φ_4)	92.51	HRL-TSCH(φ_2^{ReTx})	90.19	HRL-TSCH(φ_3)
91.23	HRL-TSCH(φ_4)	93.07	HRL-TSCH(φ_1)	93.7	HRL-TSCH(φ_1)	90.43	HRL-TSCH(φ_3)	89.18	HRL-TSCH(φ_5)
91.21	HRL-TSCH(φ_1)	91.65	HRL-TSCH(φ_6)	93.02	HRL-TSCH(φ_6)	90.03	HRL-TSCH(φ_5)	89.05	HRL-TSCH(φ_1)
91.08	HRL-TSCH(φ_5)	90.14	HRL-TSCH(φ_5^{ReTx})	92.41	HRL-TSCH(φ_2)	89.25	HRL-TSCH(φ_4)	88.89	HRL-TSCH(φ_5^{ReTx})
90.71	HRL-TSCH(φ_2^{ReTx})	90.01	HRL-TSCH(φ_5)	91.63	HRL-TSCH(φ_5^{ReTx})	89.03	HRL-TSCH(φ_1)	88.74	HRL-TSCH(φ_3^{ReTx})
90.09	HRL-TSCH(φ_6)	88.88	HRL-TSCH(φ_2^{ReTx})	91.33	QL-TSCH-3-ReTx	87.99	HRL-TSCH(φ_6)	88.12	HRL-TSCH(φ_4)
88.29	HRL-TSCH(φ_2)	87.91	HRL-TSCH(φ_2)	90.78	QL-TSCH-3	86.58	HRL-TSCH(φ_2)	87.96	QL-TSCH-3
79.03	QL-TSCH-3-ReTx	86.25	Orchestra-11	90.61	HRL-TSCH(φ_2^{ReTx})	77.11	QL-TSCH-3-ReTx	87.71	HRL-TSCH(φ_6)
76.95	QL-TSCH-3	76.85	Orchestra-11-ReTx	90.55	HRL-TSCH(φ_3^{ReTx})	73.74	QL-TSCH-3	86.26	HRL-TSCH(φ_2)
65.62	Orchestra-11	56.07	QL-TSCH-3-ReTx	80.1	QL-TSCH-5	70.28	QL-TSCH-5-ReTx	75.69	MSF-3-ReTx
55.05	MSF-3-ReTx	55.3	QL-TSCH-3	80.03	QL-TSCH-5-ReTx	69.41	QL-TSCH-5	53.9	Orchestra-11
52.46	QL-TSCH-5-ReTx	43.83	MSF-3-ReTx	79.24	Orchestra-11	43.1	Orchestra-11	41.79	Orchestra-11-ReTx
52.22	QL-TSCH-5	38.57	QL-TSCH-5-ReTx	71.71	MSF-3	30.83	Orchestra-11-ReTx	20.99	QL-TSCH-5-ReTx
43.07	Orchestra-11-ReTx	38.49	QL-TSCH-5	70.01	MSF-3-ReTx	30.66	MSF-3-ReTx	20.89	QL-TSCH-5
29.68	MSF-3	33.86	MSF-5-ReTx	69.12	MSF-5	14.33	MSF-3	20.8	MSF-3
24.78	MSF-5	11.87	MSF-3	22.81	Orchestra-11-ReTx	10.98	MSF-5-ReTx	9.91	MSF-5
14.01	MSF-5-ReTx	10.18	MSF-5	5.61	MSF-5-ReTx	9.91	MSF-5	5.61	MSF-5-ReTx

interpolation to determine these scores using the following formula:

$$s_i = s_{\min} + \frac{(s_{\max} - s_{\min}) \times (x_i - x_{\min})}{(x_{\max} - x_{\min})} \tag{22}$$

Here, s_i represents the score of the protocol, with $s_{\min}$ and $s_{\max}$ denoting the minimum and maximum scores, respectively. x_i denotes the value of the metric, while $x_{\min}$ and $x_{\max}$ represent the minimum and maximum values of the metric. The minimum and maximum scores are set to 0 and 100, or vice versa, depending on the nature of the metric. Subsequently, to calculate the final score for each protocol, we weigh the scores of each metric based on their relative importance. The final score (*S*) is computed as follows:

$$S = \sum_{i=1}^{n} w_i \times s_i$$
$$\text{where} \qquad \sum_{i=1}^{n} w_i = 1 \tag{23}$$

Here, $w_i \in W$ represents the weight of the i^{th} metric, while *n* denotes the number of metrics. The weight vector *W* comprises weights assigned to power consumption, delay, throughput, and reliability (w_1, w_2, w_3, and w_4, respectively). We consider the following weight configurations for each metric: balanced performance ($w_b = (0.25, 0.25, 0.25, 0.25)$), power efficiency ($w_p = (0.7, 0.1, 0.1, 0.1)$), delay sensitivity ($w_d = (0.1, 0.7, 0.1, 0.1)$), throughput sensitivity ($w_t = (0.1, 0.1, 0.7, 0.1)$), and reliability sensitivity ($w_r = (0.1, 0.1, 0.1, 0.7)$).

Table V displays the rankings of protocols for networks emphasizing balanced performance, power efficiency, delay sensitivity, throughput sensitivity, and reliability. These rankings are determined based on the final score assigned to each protocol, calculated using weighted scores across multiple performance metrics. In this ranking system, protocols with higher final scores are positioned at the top, signifying superior overall performance. Remarkably, the ranking remains consistent across various weight configurations, underscoring the robustness of HRL-TSCH across diverse network requirements. Notably, HRL-TSCH consistently secures the top position across all weighted scenarios, highlighting its versatility and efficacy across different network priorities. Whether the focus is on achieving a well-balanced performance or optimizing for specific criteria such as power efficiency, delay sensitivity, throughput sensitivity, or reliability, HRL-TSCH emerges as the optimal choice. This consistent dominance underscores the adaptability and reliability of HRL-TSCH in meeting the demands of modern networking applications, making it the preferred solution for a wide range of network configurations and objectives.

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced HRL-TSCH, a novel hierarchical reinforcement learning framework tailored for optimizing the TSCH schedule within an IoT network, specifically designed to meet diverse application requirements. Our approach leverages a higher-level policy for optimal link selection and a lower-level policy for optimal cell selection, both trained using the DQN algorithm. The inherent adaptability of our methodology allows it to effectively address a comprehensive range of application requirements, encompassing power consumption, delay, and throughput.

Our experiments, conducted in the Cooja network simulator, surely demonstrated that HRL-TSCH outperforms existing state-of-the-art approaches, affirming its crucial role in balancing trade-offs between power consumption, delay, and throughput.

Looking ahead, our future work will extend the optimization scope by incorporating slotframe size into the HRL-TSCH framework, recognizing the essential role of HRL

in further improving the performance of TSCH schedules. Additionally, we plan to broaden the applicability of HRL-TSCH by extending support for contention-based scheduling, solidifying HRL's key role in enhancing the versatility and adaptability of IoT networks across diverse scenarios. While our study primarily focuses on optimizing the TSCH schedule to meet specific user requirements rather than explicitly addressing single-point-of-failure issues, future research could explore strategies for enhancing system robustness, such as incorporating redundancy or fault-tolerant mechanisms. The nature of HRL in solving the TSCH scheduling problem positions HRL-TSCH as a pivotal advancement in optimizing TSCH schedules for dynamic IoT environments.

ACKNOWLEDGMENT

The document reflects only the authors' view, and the Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] S. He, K. Shi, C. Liu, B. Guo, J. Chen, and Z. Shi, "Collaborative sensing in Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 3, pp. 1435–1474, 3rd Quart., 2022.

[2] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[3] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.

[4] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Netw.*, vol. 52, no. 12, pp. 2292–2330, 2008.

[5] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomo, "TSCH and 6TiSCH for Contiki: Challenges, design and evaluation," in *Proc. 13th DCOSS*, 2017, pp. 11–18.

[6] H. Chen, X. Li, and F. Zhao, "A reinforcement learning-based sleep scheduling algorithm for desired area coverage in solar-powered wireless sensor networks," *IEEE Sensors J.*, vol. 16, no. 8, pp. 2763–2774, Apr. 2016.

[7] N. C. Luong et al., "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[8] H. Yu and K.-W. Chin, "Learning algorithms for data collection in RF-charging IIoT networks," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 88–97, Jan. 2023.

[9] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discret. Event Dyn. Syst.*, vol. 13, nos. 1–2, pp. 41–77, 2003.

[10] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–35, 2021.

[11] F. F. Jurado-Lasso, L. Marchegiani, J. F. Jurado, A. M. Abu-Mahfouz, and X. Fafoutis, "A survey on machine learning software-defined wireless sensor networks (ML-SDWSNS): Current status and major challenges," *IEEE Access*, vol. 10, pp. 23560–23592, 2022.

[12] T. Kim, L. F. Vecchietti, K. Choi, S. Lee, and D. Har, "Machine learning for advanced wireless sensor networks: A review," *IEEE Sensors J.*, vol. 21, no. 11, pp. 12379–12397, Jun. 2021.

[13] Y. Jin, P. Kulkarni, J. Wilcox, and M. Sooriyabandara, "A centralized scheduling algorithm for IEEE 802.15.4e TSCH based industrial low power wireless networks," in *Proc. IEEE WCNC*, 2016, pp. 1–6.

[14] W. Jerbi, O. Cheickhrouhou, A. Guermazi, and H. Trabelsi, "MSU-TSCH: A mobile scheduling updated algorithm for TSCH in the Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 7978–7985, Jul. 2023.

[15] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Topology management and TSCH scheduling for low-latency convergecast in in-vehicle WSNs," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 1082–1093, Feb. 2019.

[16] M. O. Ojo, S. Giordano, D. Adami, and M. Pagano, "Throughput maximizing and fair scheduling algorithms in Industrial Internet of Things networks," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3400–3410, Jun. 2019.

[17] F. F. Jurado-Lasso, M. Barzegaran, J. F. Jurado, and X. Fafoutis, "ELISE: A reinforcement learning framework to optimize the slotframe size of the TSCH protocol in IoT networks," *IEEE Syst. J.*, early access, doi: 10.1109/JSYST.2024.3371429.

[18] H. Nguyen-Duy, T. Ngo-Quynh, F. Kojima, T. Pham-Van, T. Nguyen-Duc, and S. Luongoudon, "RL-TSCH: A reinforcement learning algorithm for radio scheduling in TSCH 802.15.4e," in *Proc. ICTC*, 2019, pp. 227–231.

[19] H. Dakdouk, E. Tarazona, R. Alami, R. Féraud, G. Z. Papadopoulos, and P. Maillé, "Reinforcement learning techniques for optimized channel hopping in IEEE 802.15.4-TSCH networks," in *Proc. 21st MSWiM*, 2018, pp. 99–107.

[20] F. Veisi, J. Montavont, and F. Theoleyre, "Enabling centralized scheduling using software defined networking in industrial wireless sensor networks," *IEEE Internet Things J.*, vol. 10, no. 23, pp. 20675–20685, Dec. 2023.

[21] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "Whitelisting without collisions for centralized scheduling in wireless industrial networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5713–5721, Jun. 2019.

[22] H. Hajizadeh, M. Nabi, and K. Goossens, "Decentralized configuration of TSCH-based IoT networks for distinctive QoS: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 10, no. 19, pp. 16869–16880, Oct. 2023.

[23] H. Park, H. Kim, S.-T. Kim, and P. Mah, "Multi-agent reinforcement-learning-based time-slotted channel hopping medium access control scheduling scheme," *IEEE Access*, vol. 8, pp. 139727–139736, 2020.

[24] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. 13th Sensys*, 2015, pp. 337–350.

[25] Y. Ha and S.-H. Chung, "Traffic-aware 6TiSCH routing method for IIoT wireless networks," *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22709–22722, Nov. 2022.

[26] H. Farag, S. Grimaldi, M. Gidlund, and P. Österberg, "REA-6TiSCH: Reliable emergency-aware communication scheme for 6TiSCH networks," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1871–1882, Feb. 2021.

[27] L. Bommisetty and T. Venkatesh, "Resource allocation in time slotted channel hopping (TSCH) networks based on phasic policy gradient reinforcement learning," *Internet Things J.*, vol. 19, Aug. 2022, Art. no. 100522.

[28] A. R. Urke, Ø. Kure, and K. Ovsthus, "A survey of 802.15.4 TSCH schedulers for a standardized Industrial Internet of Things," *Sensors*, vol. 22, no. 1, p. 15, 2022.

[29] S. S. G. Shiny, S. S. Priya, and K. Murugan, "Control message quenching-based communication protocol for energy management in SDWSN," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 3188–3201, Sep. 2022.

[30] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," Comput. Syst. Lab., Swedish Inst. Comput. Sci., Stockholm, Sweden, Rep. T2011, Mar. 2011.

[31] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. Pister, "A realistic energy consumption model for TSCH networks," *IEEE Sensors J.*, vol. 14, no. 2, pp. 482–489, 2013.

[32] S. Scanzio et al., "Wireless sensor networks and TSCH: A compromise between reliability, power consumption, and latency," *IEEE Access*, vol. 8, pp. 167042–167058, 2020.

[33] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with Cooja," in *Proc. 31st IEEE LCN*, 2006, pp. 641–648.

[34] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, Jun. 2022, Art. no. 101089.

[35] T. Chang, M. Vucinic, X. Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH minimal scheduling function (MSF)," Internet Eng. Task Force, RFC 9033, May 2021.

F. Fernando Jurado-Lasso (Member, IEEE) received the B.Eng. degree in electronics engineering from the Universidad del Valle, Cali, Colombia, in 2012, and the M.Eng. degree in telecommunications engineering and the Ph.D. degree in engineering from The University of Melbourne, Melbourne, VIC, Australia, in 2015 and 2020, respectively. He is currently a Postdoctoral Researcher with the Embedded Systems Engineering section, Department of Applied Mathematics and Computer Science, Technical University of Denmark. His research interests include networked embedded systems, software-defined wireless sensor networks, machine learning, protocols, and applications for the Internet of Things.

Charalampos Orfanidis (Member, IEEE) received the Ph.D. degree in technology and health from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2020. He is currently a Postdoctoral Researcher with the Technical University of Denmark. His research interests span around low-power wide area networks, robustness, IoT, and wearables for sports and health.

J. F. Jurado received the B.Sc. degree in physics from the Universidad de Nariño, Pasto, Colombia, in 1984, and the M.Sc. and Doctoral degrees in physics from the Universidad del Valle, Cali, Colombia, in 1986 and 2000, respectively. He is currently a Professor with the Faculty of Engineering and Administration, Department of Basic Science, The Universidad Nacional de Colombia Sede Palmira, Colombia. His research interests include nanomaterials, magnetic and ionic materials, nanoelectronics, embedded systems, and the Internet of Things. He is a Senior Member of Minciencias, Colombia.

Xenofon Fafoutis (Senior Member, IEEE) received the B.Sc. degree in informatics and telecommunications from the University of Athens, Greece, in 2007, the M.Sc. degree in computer science from the University of Crete, Greece, in 2010, and the Ph.D. degree in embedded systems engineering from the Technical University of Denmark in 2014. He is currently an Associate Professor with the Embedded Systems Engineering section, Department of Applied Mathematics and Computer Science, Technical University of Denmark. His research interests primarily lie in wireless embedded systems as an enabling technology for digital health, smart cities, and the (industrial) Internet of Things.