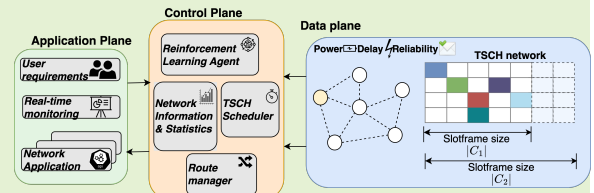


ELISE: A Reinforcement Learning Framework to Optimize the Slotframe Size of the TSCH Protocol in IoT Networks

F. Fernando Jurado-Lasso, *Member, IEEE*, Mohammadreza Barzegaran, *Member, IEEE*, J. F. Jurado, and Xenofon Fafoutis, *Senior Member, IEEE*

Abstract—The Industrial Internet of Things (IIoT) is shaping the next generation of cyber-physical systems to improve the future industry for smart cities. It has created novel and essential applications that require specific network performance to enhance the quality of services. Since network performance requirements are application-oriented, it is of paramount importance to provide tailored solutions that seamlessly manage the network resources and orchestrate the network to satisfy user requirements. In this article, we propose ELISE, a Reinforcement Learning (RL) framework to optimize the slotframe size of the Time Slotted Channel Hopping (TSCH) protocol in IIoT networks while considering the user requirements. We primarily address the problem of designing a framework that self-adapts to the optimal slotframe length that best suits the user's requirements. The framework takes care of all functionalities involved in the correct functioning of the network, while the RL agent instructs the framework with a set of actions to determine the optimal slotframe size each time the user requirements change. We evaluate the performance of ELISE through extensive analysis based on simulations and experimental evaluations on a testbed to demonstrate the efficiency of the proposed approach in adapting network resources at runtime to satisfy user requirements.

Index Terms—Smart cities, Smart resource management, Time Slotted Channel Hopping (TSCH), Reinforcement Learning (RL), Industrial Internet of Things (IIoT).



I. INTRODUCTION

THE world has witnessed a shift from traditional communication networks that interconnect computers through well-established standards to a pervasive network of networks that provides internet connectivity even to the smallest physical objects. This evolved communication network, known as the IoT, is the enabling technology for Industry 4.0, where operational technology meets information technology. These

cutting-edge technologies have created novel and essential applications for industrial operations such as smart cities, intelligent energy management, transportation, homes, waste management, etc.

At the hardware level, such applications are realized through electronic devices embedded with intelligent computing, communication systems, sensors, and actuators. The architecture of these devices, along with the need to make them flexible, cost-effective, and able to be embedded into even the smallest things, has led to the development of Wireless Sensor Network (WSN) technology. WSNs have been deployed in a wide variety of industrial applications [1]–[3]. The overall architecture, characteristics, and applications impose design constraints on their size and cost, resulting in strict resource limitations, including computation capabilities, energy, memory, and communication bandwidth. The resources of WSNs come at a significant cost; therefore, they need to be managed intelligently to ensure optimal performance for the longest possible period of time.

One of the most popular performance metrics for WSNs is energy consumption. A pioneering approach in reducing energy consumption is presented in [4], which proposes a cluster-based routing approach to extend the network lifetime. Building on this work, various research studies [5] propose

Manuscript received October 1, 2022; revised xx, xx. This work was partly supported by DAIS. DAIS (<https://dais-project.eu/>) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007273. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Spain, Portugal, Belgium, Germany, Slovenia, Czech Republic, Netherlands, Denmark, Norway, and Turkey. 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. Danish participants are supported by Innovation Fund Denmark under grant agreement No. 0228-00004A.

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

Mohammadreza Barzegaran is with the Center for Pervasive Communication and Computing, University of California, Irvine, CA 92697, USA (e-mail: barzegm1@uci.edu).

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).

new algorithms and architectures to find the best topology and routing strategies that minimize energy expenditure when transmitting packets from source to destination.

Alternatively, other approaches focus on minimizing the energy consumption of wireless sensor nodes by regularly turning off their radio chipsets [6]. However, the implementation of such sleeping periods requires strict scheduling algorithms to establish the transmitting and receiving times between neighboring nodes. Deterministic Media Access Control (MAC) protocols have been introduced to address these limitations. Scheduling algorithms orchestrate the transmissions and receptions for all sensor nodes in the WSN. Transmissions are scheduled in a way that allows non-interfering transmissions to occur simultaneously without colliding with each other. Moreover, these protocols take advantage of frequency diversity in their radios to increase network capacity and reduce interference from external devices.

The IEEE 802.15.4 - TSCH standard defines the set of functionalities required to run link scheduling for low-rate WSNs [7], [8]. The scheduling algorithm, often referred to as the scheduler, generates cyclic schedules (referred to as *slotframes*) that determine the physical channel to use for transmission at each time point.

The schedule of the TSCH network directly affects its performance. Redundant links improve link reliability and latency, but also increase power consumption as receiving nodes wake up their radios more frequently. The slotframe size represents the length of the schedule and indicates how often the schedule repeats. A smaller slotframe size improves network reliability and latency as the links repeat more frequently, but at the cost of higher power consumption.

Conversely, a larger slotframe size minimizes network power consumption but compromises network reliability and latency. Therefore, the scheduling algorithm must consider the user requirements to tailor a schedule that meets the application's needs. Additionally, the scheduling should be flexible, adaptable, and reliable to accommodate dynamic changes in the environment and user requirements.

A. Contributions

In this article, we propose an open-source reinforcement learning framework named ELISE¹ to optimize the slotframe size of TSCH networks, considering dynamic changes in user requirements. We address the problem of designing a framework that self-adapts the slotframe size of the TSCH schedule to the optimal length that best suits a set of user requirements. ELISE guides the network through a set of actions to determine the optimal slotframe size whenever user requirements change.

The main contributions of this article are as follows:

- 1) We develop a novel open-source framework that enables centralized network resource management and run-time reconfiguration of WSNs.
- 2) We develop a reinforcement learning solution that utilizes the ELISE framework to self-adapt the network's

reliability, power efficiency, and delay based on user requirements.

- 3) We design a reward model based on a multi-objective cost function that facilitates the selection of the best network configuration to meet user requirements.
- 4) We evaluate the performance of ELISE through extensive analysis using simulations and experimental evaluations on a testbed.

The remainder of this article is organized as follows. Section II provides a technical background on key concepts in the framework and an overview of related research. Section III provides a detailed description of the framework components. Section IV explains the design of the reinforcement learning framework. Section V presents the experimental layout, the approximation model of the environment to expedite the learning process of the reinforcement learning framework, the training process, and the experimental evaluation. Finally, Section VI presents the main conclusions and potential areas for future research.

II. BACKGROUND AND RELATED WORK

In this section, we briefly introduce the network model and the core technologies used throughout this paper: TSCH and Software-Defined Wireless Sensor Networks (SDWSNs). We then discuss research works that have used these technologies to improve network performance.

A. Network model

We model the network as a directed graph $G(\mathcal{V}, \mathcal{E})$, where \mathcal{V} is the set of nodes, and \mathcal{E} is the set of physical links. A node $\nu_i \in \mathcal{V}$ is the sender and/or receiver of network traffic. The number of network nodes is denoted with $|\mathcal{V}|$. A link $\epsilon_{i,j} \in \mathcal{E}$ is a full-duplex link that connects the two nodes ν_i and ν_j . since the links are bi-directional, the link $\epsilon_{i,j}$ is equivalent to $\epsilon_{j,i}$. The number of network links is denoted with $|\mathcal{E}|$.

The network traffic is modeled with the concept of streams (also called flows) that represents a data packet from one sender (talker) to one or multiple receivers (listeners). We denote the set of network streams as \mathcal{T} . A stream $\tau_i \in \mathcal{T}$ is characterized by the talker and the receiver. In ELISE, we limit the number of listeners for each stream to one, i.e., unicast communication. However, the model can be easily extended to support multicast streams by adding each sender-receiver pair as a stream.

The path for the stream τ_i is determined as an ordered sequence of directed links and denoted with $r_i \in R$. Besides, $|r_i|$ represents the number of links in the path. For example, the stream $\tau_1 \in \mathcal{T}$ sending from the node ν_1 to the node ν_3 has the route $r_1 = \{\epsilon_{1,2}, \epsilon_{2,5}, \epsilon_{5,3}\}$. Using the set of routes R , we define the function $\mathcal{H} : \nu_i \rightarrow \mathbb{N}$ which takes the node ν_i as the input and returns the node rank, i.e., the number of links originated for the node ν_i , as the output. Fig. 1 presents an example of a TSCH schedule for six-node network topology.

¹<https://github.com/fdojurado/SDWSN-controller.git>

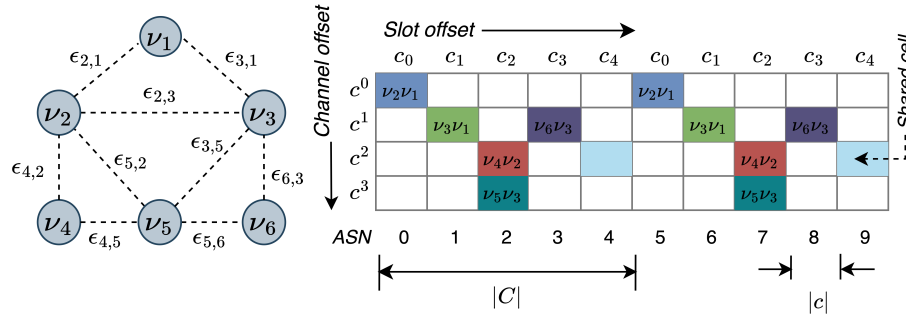


Fig. 1. An example of a TSCH schedule for a six-nodes topology.

B. SDWSNs

We model the software layer of the network using the concepts from Software-Defined Networking (SDN) and Wireless Sensor Network (WSN) that we call SDWSN. This approach has been devised as a potential pathway to solve the management and run-time reconfiguration complexities currently found in state-of-art WSNs. It adopts concepts from Software-Defined Networking (SDN). This new WSN architecture divides the control from the data functions, allowing the logically centralized controller to become reprogrammable and the WSN to be abstracted for applications and network services [9]. SDN separates the network into three network planes: application, control, and data plane. The application plane hosts applications and programs that send information about the network requirements to the SDN controller. In contrast, the control plane is a logically centralized entity that processes application requirements and sets up the network infrastructure resources to satisfy them. Lastly, the data plane is the network infrastructure with little intelligence that follows orders from the control plane. Readers interested in a thorough background, challenges, and benefits of SDWSNs can refer to [9]–[11].

C. Orchestra

Orchestra [12] is designed to run multiple stacked slotframes (static schedules) that repeat at different periods to ensure they do not interfere evenly. Each slotframe is allocated to a specific network plane that is defined by SDWSN. The scheduler selects the slotframe with higher priority to run when multiple slotframes need the communication medium simultaneously. In its default configuration, Orchestra runs three slotframes: i) Enhanced Beacon (EB), ii) unicast, and iii) default traffic slotframes. The EB slotframe is a communication link from sensor nodes to its children to set the time source. The unicast slotframe contains links to every neighbor in the WSN. The default slotframe is used for traffic other than EB and unicast packets.

D. TSCH

TSCH is a globally synchronized network where traffic is transmitted based on a static cyclic schedule table called slotframe C that repeats with the period equal to the slotframe size $|C|$ [13], [14]. In a TSCH network, the slotframe

(schedule) is divided into equal-length timeslots as shown in Fig. 1. We denote each timeslot with $c_j \in C$ and its length with $|c|$ (same length for all timeslots). The timeslot length $|c|$, typically 10 ms, is long enough for the transmission and acknowledgment of frames.

To this end, the slotframe size C is equal to the size of n timeslots, denoted with $|C| = n \times |c|$. Within a slotframe, timeslots are counted with their subscripts j . Similarly, the timeslots can be counted from when the network booted with an Absolute Slot Number (ASN). The ASN serves as a global clock, and it increases at every timeslot.

Each timeslot c_j is divided into a fixed number of cells. Each cell denoted with c_j^k indicates the channel offset k . For channel hopping a cell is used to find the physical channel to transmit. Considering an array of channel frequencies F to hop over, the channel frequency f for the cell c_j^k is calculated in Eq. (1).

$$f = F[(j + k)\%|F|] \quad (1)$$

To this end, each cell will select a different physical channel at consecutive slotframes. The scheduler oversees the role of cells in the slotframe. The roles can be shared, dedicated, or empty cells. Shared cells (light blue cells in the example) are contention-based. They are used by multiple transmitters, increasing the probability of interference as they can transmit simultaneously. Reliable transmissions resend frames using a back-off window when no acknowledgment is received. Dedicated cells (cells labeled with a pair of nodes) are contention-free. They are allocated carefully not to cause interference issues with other cells. Retransmissions can occur due to external interference or bad radio link quality [15].

The example provides an illustration of a TSCH schedule that has been designed for a six-node WSN. The slotframe size is five timeslots $n = 5$. There are six cells with a non-empty role, one shared cell shown with c_4^2 . There are five dedicated cells that permit the communication of nodes with their parents. There are also two non-interfering transmissions at the same slot offset (c_2^2, c_3^2).

E. Related work

Table I presents a summary of the latest research works on related topics including TSCH, SDWSNs, and RL. It also provides information related to the year of conception,

TABLE I
SUMMARY OF RELATED WORKS.

Article	Year	Topic			UR	Major contribution(s)
		TSCH	SDWSNs	RL		
[12]	2015	✓	✗	✗	✗	Autonomous scheduler for TSCH without control overhead that does not rely on centralized or distributed entities.
[16]	2017	✗	✓	✗	✓	A decision-making approach to select the routing protocol that best suits the application requirements to get optimal performance using supervised learning.
[17]	2020	✓	✓	✗	✗	An SDN-based network architecture that provides support to mobile nodes using TSCH. Mobile nodes have one up- and down-link to every node in the network regardless of their position in the WSN.
[18]	2020	✗	✓	✗	✗	An SDN-based approach to pinpoint mobile nodes in WSNs. The approach features a mobility detector and a k-means cluster algorithm to decouple static from mobile nodes.
[15]	2020	✓	✗	✗	✗	A low-latency distributed scheduling function to optimize the End-to-End delay, and reliability.
[19]	2021	✗	✓	✓	✗	A traffic monitoring framework for SDWSNs. They trained a double deep Q-network (DDQN) agent to achieve the optimal flow rule match-field policy.
[20]	2021	✗	✓	✓	✗	A reinforcement learning approach to select the best routing path that improves the QoS of the SDWSN.
[21]	2023	✓	✗	✓	✓	A reinforcement learning approach to configure the parameters of the CSMA/CA of the TSCH protocol to achieve distinctive QoS.
ELISE	2023	✓	✓	✓	✓	An open-source framework that self-configures network resources in run-time to satisfy the dynamic user requirements for SDN-based IoT networks.

whether the given article considers user requirements, and major contributions. Research works [17], [18] strive to improve the network performance using SDWSNs. The centralized architecture of SDWSNs allows the control plane to build a global view of the network that permits it to make better decisions, in this case, to provide support for mobile sensor nodes. The research work in [17] uses a static TSCH schedule with redundant links for mobile nodes to improve reliability, whereas [18] uses a supervised learning approach to separate mobile from static nodes. Articles [19], [20] oversee the IoT network through the centralized controller, which enables the collection of observations to design, implement and train a learning agent to maximize the accumulative reward of actions taken. The work in [19] focuses on monitoring the SDWSN traffic, at granularity levels to mitigate flow-table overflows, while [20] uses a learning agent to find the optimal forwarding paths for the SDWSN. Articles [12], [15] aim to improve the performance of TSCH networks. Research work [15] objective is to reduce the packet latency. This is achieved by dividing the slotframe into small chunks. Sensor nodes select the chunk to transmit based on their distance to the border router to minimize the latency. The work in [12], which has been previously introduced, is an autonomous scheduler with little overhead that provides high reliability. The article in [16] main objective is to select among a set of routing algorithms the best that suits the given user requirements. The selection of the routing algorithm is achieved using a supervised learning approach. The research work in [21] presents a RL approach to configure the Carrier-Sense Multiple Access/Collision Avoidance (CSMA/CA) parameters in a multi-hop TSCH network. The framework utilizes neural networks to converge to Quality of Service (QoS) satisfying configurations efficiently.

Overall, these studies aim to enhance the QoS in network applications. Performance metrics such as energy, delay, and reliability are considered, but there has been limited attention given to real user needs. It is crucial to provide a customized engineering solution that seamlessly manages network

resources and orchestrates the network to meet the specific requirements of applications or users. However, as observed from the table, this aspect has not received significant focus. While the work presented in [16] takes user requirements into account in their proposed approach, their supervised learning method encounters challenges in predicting the optimal routing algorithm in dynamic environments like WSNs. Furthermore, they do not consider the MAC layer, which is responsible for the duty cycle of sensor nodes in state-of-the-art WSNs, directly impacting power consumption, delay, and reliability.

III. ELISE FRAMEWORK ARCHITECTURE OVERVIEW

This section presents an overview of the ELISE functional framework and architecture. Next, a detailed description of their components is presented, such as the layered architecture planes.

A high-level overview of the ELISE framework architectural structure, including its main components and interfaces, is shown in Fig. 2. The architectural structure follows the typical three-tier SDN principles for WSNs. The description of each layer of the architecture from the bottom-up is as follows.

A. Data plane

The data plane, also called the infrastructure plane, is built upon the interconnection of multiple Networked Embedded Systems (NES). NES, also named sensor nodes or network nodes, are resource-constrained devices embedded with a processing unit, a memory unit, a communication transceiver, and some form of power supply. They are mainly programmed for a specific task, such as monitoring a physical variable and are often deployed in harsh environments. In ELISE, we have defined two types of NES: the (regular) sensor node and the sink. All sensor nodes, including the sink, communicate with each other using IEEE 802.15.4 radios; however, the sink is also directly connected to the control plane through a wired interface. All NES are nodes of the network graph that are denoted with $\nu \in \mathcal{V}$ and all radio communication links are

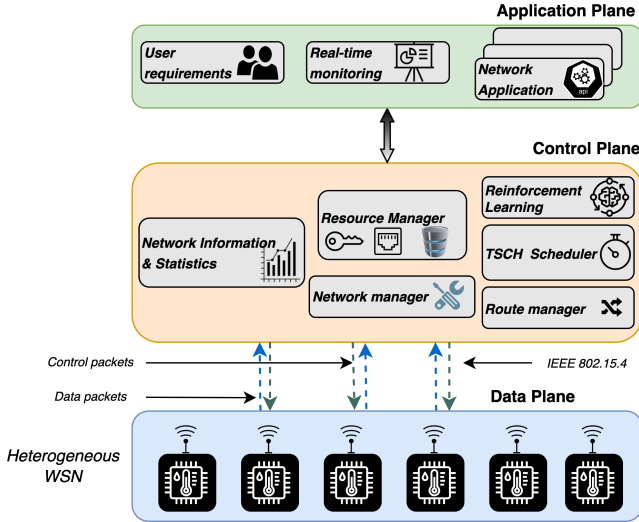


Fig. 2. ELISE architecture.

denoted with $\epsilon \in \mathcal{E}$, see Sect. II-A for more information. Besides, we assume that sensor nodes are the talkers of network streams and all streams have the same listener node, i.e., the sink or the controller node.

Overall the entire network infrastructure runs on a lightweight embedded operating system. Among the available embedded operating systems in the market [9], we have selected Contiki-NG [22] because

- 1) it is open-source, well documented, and it has a large community,
- 2) it is widely used in the research community,
- 3) it provides the implementation of TSCH and Orchestra,
- 4) it can run on both Cooja network simulator [23] and real hardware.

To comply with SDWSN principles of making the network infrastructure run simple tasks and remove energy-intensive functions from sensor nodes, we have redesigned the protocol stack, from layers three and up, to support the following five functionalities.

1) Data packets: This packet encapsulates the collected data and sends them to the control plane. The packet format is shown in Fig. 3. The cycle sequence and sequence fields are used, by the reinforcement learning algorithm, to keep track of the number of packets received in the corresponding cycle. Temperature, humidity, and light are physical variables measured by sensors (this can be generalized to variables one, two, and three). The ASN field contains the ASN when the packet was created. This field is useful to calculate the packet latency under specific network configurations such as routes, TSCH schedules, etc.

2) Neighbor discovery (ND): This packet discovers other sensor devices in the sender transmission range. It also allows discovering neighbors with paths to the controller. This packet contains three fields: rank, Received Signal Strength Indicator (RSSI), and checksum. The rank field is equivalent to \mathcal{H} which is the number of links originated from the talker node (see Sect. II-A). The RSSI field specifies the accumulative RSSI to the controller. This field permits the receiver node to decide

Data Packet																																					
Offset	Byte	0							1							2							3														
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7				
0	0	Cycle seq														Seq							Temperature														
4	32	Humidity							Light							Absolute																					
8	64	Slot Number (ASN)																																			

Fig. 3. Data packet format.

Neighbour Advertisement (NA)																																				
Offset	Byte	0							1							2							3													
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7			
0	0	Payload Length							Sender Rank							Sender Power																				
4	32	Cycle Sequence														Seq							Padding													
8	64	CRC of the entire packet																																		

Fig. 4. Neighbor advertisement header format.

Packet format for a specific coordinate																																	
Offset	Byte	0							1							2							3										
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	Type (Rx, Tx)							Channel offset							Time offset							Padding										
4	32	Source Address														Destination Address (For RX=NULL)																	

Fig. 5. Packet format of a TSCH schedule link.

which parent to choose between two equal rank values. Lastly, the checksum field is an error checking of the packet integrity.

3) Neighbor advertisement (NA): This packet contains messages to report their status and neighbors' to the controller, including the average power consumption, rank, and links to neighbors. The format of the packet header is shown in Fig. 4. The payload length field states the number of bytes contained in the packet payload. The sender rank specifies the rank of the sender. The sender power field contains the power consumption of the sender. Cycle sequence and sequence fields fulfill the same function as in the data packet. The CRC field is an error checking of the packet integrity. The payload consists of neighbors' addresses, RSSI, and Link Quality (LQ) values.

4) Network configuration - TSCH schedules: This packet type is a control message to establish the TSCH schedules for the incoming cycle. The header has four fields: payload length, slotframe size $|C|$, sequence, and CRC. The payload length, sequence, and CRC fields fulfill the same functions mentioned above. The slotframe size field contains the length of the schedules encapsulated in the payload. The format of a TSCH link embedded in the payload of this control packet is shown in Fig. 5. The type field states the type of TSCH link: transmit (Tx) or listen (Rx). The channel and time offset fields specify the coordinates of the given link. The source address field indicates the address of the sensor node that needs to process this link. Lastly, the destination address is used for Tx link types to set the neighbor address.

5) Network configuration - Routes: This packet type is also a control message to establish the forwarding paths for the incoming cycle. The packet header consists of payload length, sequence, and CRC fields which fulfill the same function mentioned above. The packet payload contains the source, destination, and neighbor addresses to build the forwarding paths.

Control packets including TSCH schedules, and route packets are broadcasted into the WSN. ELISE takes advantage of the implementation of TSCH and Orchestra in Contiki-NG to devise a novel approach that enables a run-time reconfiguration and distribution of TSCH schedules and route packets. ELISE defines four slotframes, inspired by Orchestra, for specific traffic planes: EB slotframe for the time source, control traffic slotframe for control packets, data traffic slotframe for data packets, and the default traffic slotframe for any other type of traffic. The control traffic is a broadcast slotframe that permits the transmission and reception of new TSCH schedules and route configurations.

For cell c_j^k , the time slot number j and channel offset k for sensor node ν_i and control plane slotframe size of $|C|$ are calculated as follows.

$$j = \mathcal{H}(\nu_i) \% |C| \quad (2)$$

$$k = ((i) \% |C| - 1) + 1 \quad (3)$$

These two equations minimize communication interference between sensor nodes with equal rank values that try to broadcast control packets to their children.

B. Control plane

The network intelligence resides in this plane. As seen in Fig. 2, the control plane hosts multiple modules for the correct functioning of the framework. The control plane can run locally on a computer or in the cloud. At its core, it is implemented in Python 3 [24] because it supports multiple machine learning libraries available in the market, has a large community, and has an ample library collection. Here, we briefly introduce the core functionalities of each module.

1) *Network information and statistics*: This module holds all network information collected. It also includes all packets received and simple statistics.

2) *Resource manager*: This module is in charge of orchestrating all resources in the control plane. Resources in this module include database, serial interface, and network access.

3) *Network manager*: This module holds key functions to correctly operate the data plane. Functions such as writing and reading from the network reside here.

4) *Route manager*: This module hosts all functionalities to build the forwarding paths of the network. It hosts multiple traditional routing algorithms. It is also flexible to allow adding new centralized routing protocols.

5) *TSCH manager*: In this module resides the functions to correctly build TSCH schedules. It has been designed to easily create new TSCH schedulers on top.

6) *Reinforcement learning*: The reinforcement learning module is the main intelligent component of the entire control plane. It uses all other modules to collect data, learn from the environment, tune hyper-parameters and evaluate the trained agent. This module is discussed in detail in the next section.

C. Application plane

The application plane hosts user requirements and programs such as real-time monitoring tools that convey information

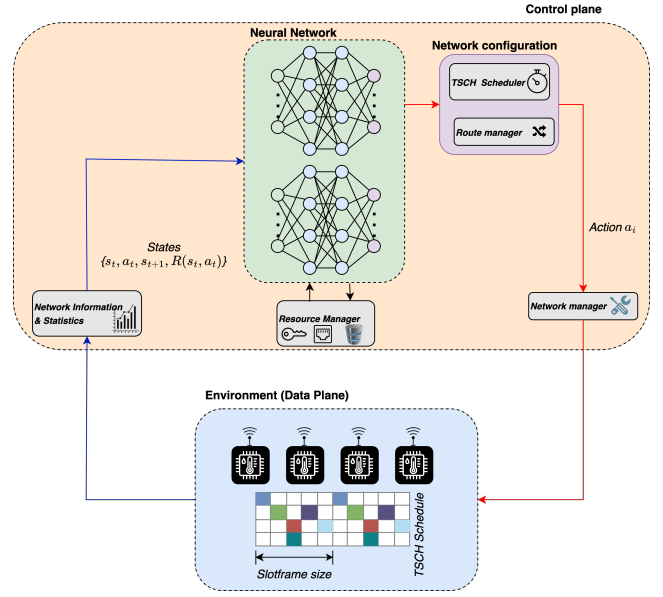


Fig. 6. The architecture of the RL agent of ELISE.

regarding the status of the network. This plane instructs the control plane on the current user requirements through the northbound API.

IV. REINFORCEMENT LEARNING MODULE

This section takes the reader through the design and implementation of a reinforcement learning approach to optimize the slotframe size of TSCH in SDN-based IoT networks considering the user requirements. Readers interested in the background of RL can refer to [25] and its applications to SDWSNs can be found in [9].

A. Solution approach

To solve this multi-objective function that self-adapts the slotframe size of the TSCH schedule to the optimal length that best suits a set of user requirements, we use a reinforcement learning agent that hosts neural networks in its core, as shown in Fig. 6. ELISE provides flexibility to evaluate different reinforcement learning algorithms. Algorithms that we consider in this research are Deep Q Network (DQN) [19], Asynchronous Advantage Actor Critic (A2C) [26], and Proximal Policy Optimization (PPO) [27].

The proposed solution follows the typical three-tier principles for SDWSNs, where the control plane layer collects data, orchestrates resources, performs intelligent calculations, and deploys new network configurations into sensor nodes. At the initial state of the data plane, sensor nodes discover their path to the controller using ND packets. They then start sending NA packets to the controller. The controller processes these packets to make future decisions. The reinforcement learning agent predicts the next slotframe size. It then prepares the TSCH schedules and routes using the TSCH and route manager module. Finally, the control plane deploys new configurations through the network management module.

ELISE develops a Markov Decision Process (MDP) framework with the architecture depicted in Fig. 6. This framework enables the reinforcement learning algorithm to dynamically select and deploy optimal actions based on the observations to maximize the average accumulative reward. The MDP is represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, where \mathcal{S} represents the state space, \mathcal{A} represents the action space, and \mathcal{R} represents the immediate reward.

- **State Space:** As previously discussed, there are three performance metrics, and three user requirements, at the end of each iteration. However, the learning time can be reduced by adding the last scheduled link in the TSCH slotframe (λ), and the current slotframe size ($|C|$). λ enables the agent to avoid slotframe sizes that are below the last scheduled link in the scheduler, otherwise, it can alter the normal behavior of the TSCH network. The state space of the proposed work is defined as follows.

$$\mathcal{S} \triangleq \{(\Omega_1, \alpha_1, \beta_1, \gamma_1, \lambda_1, |C|_1), \dots, (\Omega_n, \alpha_n, \beta_n, \gamma_n, \lambda_n, |C|_n)\} \quad (4)$$

Where Ω represents the cost of the SDWSN. α , β , and γ are the user-defined coefficients for power consumption, delay, and reliability, respectively.

- **Action Space:** The RL agent aims to find the optimal slotframe size of the data plane traffic plane given the set of user requirements. At every decision-making point, the agent predicts the next slotframe size given the above-mentioned observations. The agent can take multiple consecutive actions (slotframe sizes) before reaching the optimal solution. The number of steps taken to reach the optimal solution depends on the current state of the environment, especially, the current slotframe size ($|C|$) and the user requirements (α , β , γ).

The agent selects the next action between

- 1) increasing $|C|$,
- 2) decreasing $|C|$, or
- 3) continuing using the current $|C|$.

The selection of the slotframe size is bounded by numbers that are mutually prime to other slotframes in the TSCH network. Recall that the TSCH network runs multiple stacked slotframes that repeat at different periods (mutually prime slotframe sizes) to ensure they do not interfere evenly. Therefore, the action space for selecting the next slotframe size in the control plane is defined as follows.

$$\mathcal{A} \triangleq \{a : \gcd(a, |C|^{EB}, |C|^{CP}, |C|^{DF}) = 1\} \quad (5)$$

Where \gcd represents the greatest common divisor, and $|C|^{EB}$, $|C|^{CP}$, and $|C|^{DF}$ are the slotframe sizes of the EB, control, and default traffic planes.

- **Immediate reward function:** The reward function has been designed to select slotframe sizes within a valid range and to ease learning. Whenever the agent selects a slotframe size below λ , it is penalized. In the other case, whenever the agent selects a slotframe size that goes beyond the maximum valid slotframe size (μ), it is also penalized. The agent learns to select the next action within this valid range while maximizing the accumulative reward.

The agent is positively rewarded if the slotframe size lies within the valid range. The amount rewarded depends on the performance metrics and user requirements. The reward function is expressed as follows.

$$\mathcal{R}(s, a) = \begin{cases} -\mathcal{G}_{max}, & |C|^{DP} \geq \mu \\ \Upsilon - \Omega, & \lambda < |C|^{DP} < \mu \end{cases} \quad (6)$$

Where $|C|^{DP}$ is the slotframe size of the data traffic slotframe, and \mathcal{G}_{max} is the maximum penalty for taking an invalid slotframe size. Υ is a constant that makes sure the immediate reward stays always positive. Υ is equal to the worst case of Ω . Therefore, $\Upsilon = 2$. It is noteworthy that we changed the signs in (7) to maximize the immediate reward function. Also, we have defined two terminating conditions for episodes. We end an episode either every time the agent selects a slotframe size outside the valid range or when we reach the maximum number of timesteps.

B. Cost Function

As discussed in previous sections, it is vital to take into consideration the dynamic changes in user requirements to design a tailored engineering solution that self-adapts to these changes. This approach will allow the framework to self-reconfigure the infrastructure resource that complies with the current state of user requirements while maximizing network performance. This reinforcement learning-based model optimizes the slotframe size of the data plane slotframe introduced in the above sections. Other slotframes are not considered in the optimization because they are control slotframes that are mandatory for the basic operation of the network, and tempering with them can make the network fail.

We now design the overall objective cost function as a combination of several individual objective metrics, capturing the power consumption, stream delay, network reliability, and network dependability forming a multi-objective cost function. Since our main objective is to maximize the network performance given a set of user requirements, we design a weight-based multi-objective function. Specifically, the weights of this function are the set of user requirements. This multi-objective cost function can be expressed as follows.

$$\begin{aligned} \Omega &= \omega_1 \times \omega_2 \\ \omega_1 &= \alpha \times \tilde{P} + \beta \times \tilde{D} + \tilde{R} \times (1 - \gamma) \\ &\text{subject to } \alpha + \beta + \gamma = 1. \end{aligned} \quad (7)$$

where ω_1 captures the cost of scheduling and ω_2 captures the cost of routing, both presented in Sect. IV-C. For the cost of scheduling, α , β , and γ are the user requirements for power consumption, delay, and reliability, respectively (see Sect. IV-C for more information). Since we aim at minimizing the cost function, the inverse of the reliability is used which helps to find the maximum reliability cases. They are set by the users, and the overall summation is equal to the unity. ELISE calculates the performance metrics at the end of every cycle; therefore, the samples within the time interval are considered

a constraint (they are strictly greater than zero). This holds for all performance metrics samples. The slot duration of TSCH networks can not be less or equal to zero. It is worth mentioning that the optimization is done over the slotframe size of the data plane slotframe of the TSCH protocol. Therefore, the SDN control plane takes several sequential actions at each network state to find the optimal slotframe size given the user requirements and current network state.

C. Objectives

This section gives the details of the terms of the cost function Ω that captures the network performance. The performance metrics are as follows: power consumption, delay, reliability, and dependability. The first three metrics are used in the cost term ω_1 and the dependability is defined with ω_2 .

1) Power consumption: The power consumption is collected from each network node in the WSN. It mainly depends on the energy spent to transmit and receive a packet. In TSCH networks, nodes wake up their radios at specific timeslots to either transmit a packet or listen to the wireless medium, then switch to another state such as low power mode. In ELISE, we assume three states for a network node: i) listening, ii) transmitting iii) listening and transmitting (forwarding). We ignore the idle state of the nodes since we aim at minimizing the workload of the network nodes.

The power consumed for transmitting the network stream $\tau_i \in \mathcal{T}$ from the talker to the listener denoted with P_i is calculated from Eq. (8). The consumed power is normalized in the range $[0, 1]$.

$$P_i = \frac{\sum_{\epsilon_{m,n} \in \tau_i} (V^m \times I_{tx}^m + V^n \times I_{rx}^n) \times |c|}{(V^m \times I_{max}^m + V^n \times I_{max}^n) \times |C|} \quad (8)$$

where V^i is the power supply voltage of the node ν_i (in volts), I_{tx}^i is the transmission current consumption of the node ν_i and I_{rx}^i is the reception current consumption of the node ν_i (both in mA). The maximum value of the current consumption while transmitting and receiving is captured by I_{max} . The average power consumption of the network \bar{P} is defined as the average of the power consumed for transmitting all network streams. The controller receives the NA packets from network nodes and processes the data in the sender power fields of the packets and stores them in the database. When a cycle finishes, the controller retrieves, from the database, the latest power consumption, and calculates the network power consumption cost \bar{P} using Eq. (9).

$$\begin{aligned} \tilde{P}(\theta_1) &= \bar{P} + \theta_1 \times \sigma_P \\ \bar{P} &= \frac{1}{|\mathcal{T}|} \sum_{\tau_i \in \mathcal{T}} P_i \\ \sigma_P &= \sqrt{\frac{\sum_{\tau_i \in \mathcal{T}} |P_i - \bar{P}|}{|\mathcal{T}|}} \end{aligned} \quad (9)$$

where \bar{P} is the average power consumption in the network, σ_P is the normal distribution of the node power consumption in the network, and θ_1 is the weight of the node power

consumption distribution. The larger θ_1 value drives the search for a solution with evenly distributed power consumption across the nodes. With θ_1 , the user achieves the desired distribution based on the requirements.

2) Delay: The packet delay is calculated as the interval from when the packet is generated at the talker to when the packet is received by the listener in the control plane. For a TSCH network, we can estimate the packet delay of the stream τ_i using the ASN as follows. The packet delay value is normalized in the range of $[0, 1]$.

$$D_i = (ASN_{tx} - ASN_{rx}) \times \frac{|c|}{|C|} \quad (10)$$

The control plane then calculates the average delay of the network \bar{D} using Eq. (11) at the end of each cycle. The controller retrieves from the database, the latest \bar{D} for all network streams $\tau_i \in \mathcal{T}$.

$$\begin{aligned} \tilde{D}(\theta_2) &= \bar{D} + \theta_2 \times \sigma_D \\ \bar{D} &= \frac{1}{|\mathcal{T}|} \sum_{\tau_i \in \mathcal{T}} D_i \\ \sigma_D &= \sqrt{\frac{\sum_{\tau_i \in \mathcal{T}} |D_i - \bar{D}|}{|\mathcal{T}|}} \end{aligned} \quad (11)$$

where σ_D captures the normal distribution of stream delays. Using θ_2 user can adjust the weight of delay distribution in calculating the average delay. The larger weight drives the search for a solution with evenly distributed stream delays. Similar to θ_1 , the user achieves the desired distribution using θ_2 set.

3) Reliability: To calculate the reliability, we first define the Packet Delivery Ratio (PDR) during a cycle in Eq. (12). The control plane performs the calculation of the PDR as follows.

$$\begin{aligned} PDR &= \frac{|\tau_{rx}|}{|\tau_{tx}|} \\ \text{subject to } \tau_{tx} &> 0. \end{aligned} \quad (12)$$

where τ_{rx} and τ_{tx} are the numbers of received and transmitted packets, respectively. The control plane queries the database to obtain the latest PDR values in the network and calculates the network reliability \bar{R} at every end of a cycle as follows.

$$\begin{aligned} \tilde{R} &= \frac{1}{PDR} \\ \text{subject to } PDR &> 0. \end{aligned} \quad (13)$$

4) Dependability: We define the dependability metric ω_2 as the cost function for finding the best path for the streams in the network. The dependability metric is based on a simple concept, i.e., the network is more dependable where fewer nodes are involved in the network functioning (stream transmission). Based on this concept, the most dependable network is the one where all streams use the shortest path for transmission.

To this end, we define the power dependability $PDEP$ of the node $\nu_i \in \mathcal{V}$ in Eq. (14). The power dependability of a node shows how much the node is under the workload, i.e., transmission and receiving the data. The more a node is under the workload, the less dependable is the node.

$$\begin{aligned}
DEP_i &= DEP_i^{tx} + DEP_i^{rx} \\
DEP_i^{tx} &= \sum (V^i \times I_{tx}^i \times |c|) \\
\text{subject to } &\forall \epsilon_{i,x} \in r_y \text{ and } \forall r_y \in R, \\
DEP_i^{rx} &= \sum (V^i \times I_{rx}^i \times |c|) \\
\text{subject to } &\forall \epsilon_{z,i} \in r_w \text{ and } \forall r_w \in R
\end{aligned} \tag{14}$$

The power dependability of the node represents the workload of the node which is used for determining the most dependable path for a specific stream. Thus, we calculate the dependability of the stream $\tau_j \in \mathcal{T}$ as follows.

$$\begin{aligned}
DEP_j &= \prod PDEP_i \\
\text{subject to } &\epsilon_{i,x} \in r_j.
\end{aligned} \tag{15}$$

With the dependability of the streams in the network, we calculate the average dependability of the network ω_2 using Eq. (16).

$$\begin{aligned}
\omega_2 &= \frac{1}{|\mathcal{T}|} \sum_{\tau_j \in \mathcal{T}} DEP_j \\
\text{subject to } &|\mathcal{T}| > 0.
\end{aligned} \tag{16}$$

V. PERFORMANCE EVALUATION

This section tests ELISE's ability to self-adapt the network resources to the configuration that best satisfies the dynamic user requirements. Multiple user requirements are put in place to get insights into the impacts on the performance metrics. We compare the performance of ELISE against Orchestra, which is the TSCH scheduler of choice for IoT networks. Experiments are conducted in both the Contiki Cooja network simulator and a real-world testbed; however, simulation results that have confirmed the same findings as the testbed experiments are omitted due to space constraints.

A. Testbed setup

The experiments are conducted on the premises of the FIT IoT Lab [28]. This testbed, which has six different sites across France, offers facilities with hundreds of wireless sensor nodes that allow the evaluation and experimentation of large-scale WSNs. Processors architectures supported include MSP430, STM32, Cortex-A8, and 802.15.4 radios chips running at 800 MHz or 2.4 GHz. They also provide a CLI tool to access the testbed to manage resources and experiments. It also supports a range of embedded operating systems.

We specifically built a 10-sensor-node network ($|\mathcal{V}| = 10$) with a maximum depth of three hops. The topology is shown in Fig. 7. We use this network to not overcomplicate the experiments and to let us draw conclusions from the proof of concept of the framework. We use the IoT-LAB M3 platform for nodes and the sink. This platform has embedded an ARM-Cortex M3 microcontroller (32-bit CPU @72 MHz), an ATMEL radio running at 2.4 GHz (designed for the IEEE 802.15.4 standard), and four sensors (light, pressure & temperature, accelerometer, and gyroscope). This network operates with the Contiki-NG operating system. As aforementioned, we have redesigned the

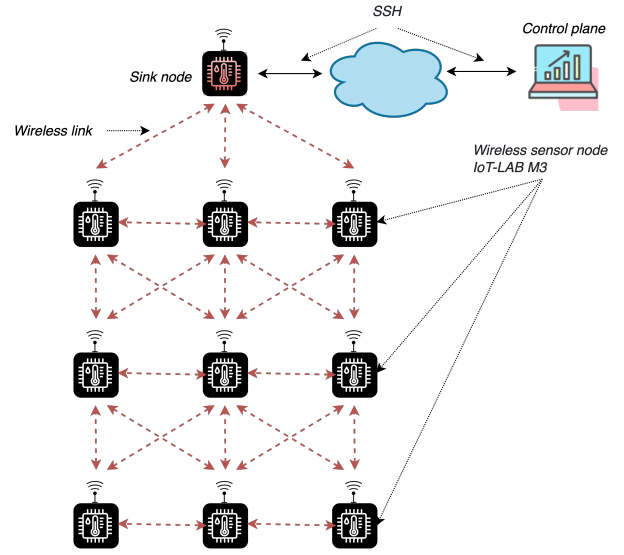


Fig. 7. Topology of the testbed setup.

Contiki-NG network stack to support the SDWSN functionalities described in Section III-A and IV, including support for all Orchestra-based slotframes. The control plane communicates with the data plane, specifically the sink node, through the Secure Shell Protocol (SSH). The control collects from and transmits commands to the WSN using this protocol. It is written in Python 3.10, uses Stable Baselines 3 [29] for the RL package, and implements all functionalities previously described in Section III-B and IV. Besides, it operates on a remote computer running macOS Big Sur on an i9 processor of eight cores at 2.3 GHz. The experiment parameters used throughout the results are summarized in Table II. There exist a trade-off between the value for the iteration window interval. A large window interval will filter out noise in the collected data; however, the control plane may not be able to react to dynamic changes in the network. On the other hand, if the interval is small, the number of network reconfigurations increases and so does the noise. This is an open research question in SDWSNs [16]. In our experiments we set this value to 60 control packets ($|\mathcal{T}| = 60$); however, this value can be changed to meet user requirements. Although with θ_1 and θ_2 in objective metrics (see Sect. IV-C) the desired distribution of power and delay is achievable, for the evaluation where we have considered a fixed path for streams, we set both θ_1 and θ_2 to zero which remove the effect of power and delay distribution in the search. In future work, ELISE will determine the path for streams.

Before we jump into the evaluation of ELISE, we want to discuss in the next section the challenges found while training the reinforcement learning agent.

B. SDWSN model approximation

The training of the learning model involves taking the collected experience to adjust the weights of the deep neural network. The collected experience is a group of pair state-actions. A group of a pair of state-action is formed at the end of every action taken. The framework selects and deploys

TABLE II
EXPERIMENT PARAMETERS.

Parameter	Value
No. of sensor nodes ($ \mathcal{V} $)	10
Sensor nodes type	IoT-LAB M3
Transmission power	-17dbm
Iteration window interval	60pkts
Power measure sample time (t_{sample})	60s
Sensor node supply voltage (V)	3V
Network node rank (\mathcal{H})	3
TSCH slot duration ($ c $)	10ms
Reward constant Υ	2
Minimum slotframe size λ	10
Maximum slotframe size μ	70
Penalty \mathcal{G}_{max}	-4
Max. episode length (\mathcal{V})	50

an action. It then waits for the cycle to finish to obtain the observations given that action. The time to complete an iteration impacts the total training time of the model. Training the model in the testbed is not feasible as an iteration can take a couple of minutes to complete depending on the frequency of control packets (NA packets) and the window interval. Although the Cooja network simulator reduces the iteration time to tens of seconds, it is not enough to train the network in a reasonable amount of time. The total training time can be estimated by $iter * ts$, where $iter$ is the average time of an iteration, and ts is the total number of timesteps e.g. if Cooja takes 30 seconds to complete an iteration then it can take up to five weeks to train for 100k timesteps. For real-world deployment an iteration can take three minutes to complete, depending on the window interval; therefore, it can take up to 3.4 months to train for 50k timesteps. To solve this issue, we mathematically model the TSCH network in the function of the slotframe size for the data plane traffic. This mathematical model allows us to estimate the network cost given a slotframe size. These performance metrics are needed for the immediate reward calculation previously discussed in Section IV-A. This approach reduces the time of an iteration significantly. This approximation model is also useful for hyperparameter tuning, which searches for the best model architecture by creating multiple scenarios with different hyperparameters using pruning strategies.

The main objective of the SDWSN approximation model is to estimate the values of the network average power consumption \tilde{P} , the average network delay \tilde{D} , and the network reliability \tilde{R} when changing the slotframe size $|C|$. It facilitates the calculation of the immediate reward of an action taken. These values are estimated using the minimum mean square error (MMSE) estimator ($E = \sum_{j=0}^k |p(x_j) - y_i|^2$). To obtain them in the function of the slotframe size, we program a simple task in the control plane. It builds and sends TSCH schedules with different slotframe sizes using the TSCH scheduler and network manager of the proposed architecture (see Fig. 2). The controller continuously selects and sends a slotframe size $|C|$ from the set of slotframe size numbers that are mutually prime to other slotframes. We then plot the values for \tilde{P} , \tilde{D} , and \tilde{R} using the 95% confidence interval. We then find the vector coefficients(ζ) that minimize the squared error in the degree

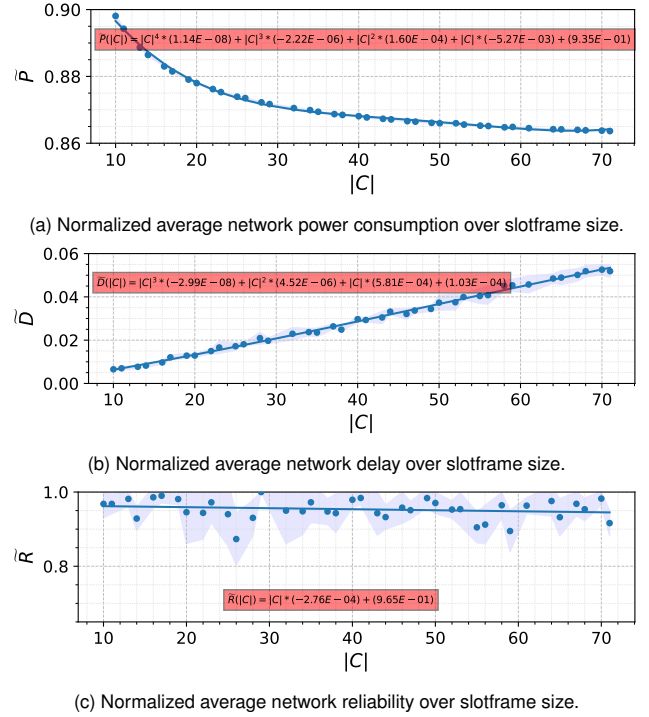


Fig. 8. Experimental \tilde{P} , \tilde{D} , and \tilde{R} values for different slotframe sizes $|C|$.

order of four, three, and one for \tilde{P} , \tilde{D} , and \tilde{R} , respectively.

We use the testbed setup shown in Fig. 7, and the experiment parameters in Table II. The plot charts in Fig. 8 show the estimated normalized values of the network average power consumption (\tilde{P}), delay (\tilde{D}), and reliability (\tilde{R}) against the slotframe size ($|C|$). The \tilde{P} metric decreases exponentially as $|C|$ increases as shown in Fig. 8a. This is expected as when we increase the slotframe size, we increase the number of unused timeslots in the slotframe. Therefore, it reduces the average power consumption in sensor nodes. Fig. 8b shows that the \tilde{D} linearly increases with $|C|$. This is also expected since the time between links in consecutive slotframes increases, therefore, packets wait longer in the queue to be transmitted. Fig. 8c shows that the \tilde{R} linearly decreases with $|C|$ but at a smaller rate. Links in TSCH networks are very reliable due to their time and frequency diversity; however, they can decay when using a large slotframe size due to the increasing waiting time for retransmissions and congested packet queues. We now use this approximation model to train the DQN, A2C, and PPO algorithms and put them under test in the testbed.

C. Training

Fig. 9 present the learning process of DQN, A2C, and PPO reinforcement learning algorithms running at the RL module of the control plane. It is noteworthy that every episode starts at a random state, this includes random slotframe size. Training for diverse starting slotframe sizes permits the agent to learn how to solve the problem in the presence of multiple states, e.g. new user requirements. Points in the chart represent the average accumulative reward during the last 1000 iterations. It is clear that the convergence performance of PPO is better

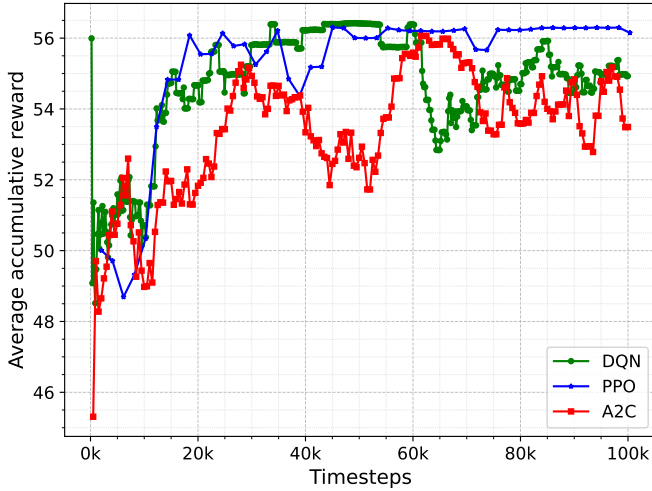


Fig. 9. Learning process of DQN, A2C, and PPO for solving the overall objective function.

TABLE III

EVALUATION OF DQN, A2C, AND PPO ALGORITHMS OVER 100 EPISODES.

Algorithm	Avg. Accumulative Reward
PPO	56.44 ± 0.11
A2C	56.37 ± 0.14
DQN	56.34 ± 0.15

than that of the DQN and significantly better than A2C. PPO converges after approximately 50k iterations, and the average accumulative reward is roughly 56.3. Although DQN reaches the maximum average accumulative compensation at approximately 50k iterations, it only converges to an average accumulative reward of 55 after 80k iterations. In contrast, A2C has an average accumulative reward of approximately 54, but it is still struggling to converge to a steady value. PPO seems the best candidate to solve the problem; however, we also evaluate the algorithms' performance by taking solely deterministic actions over 100 episodes to decide on which algorithm to pick. Table III shows that PPO obtained the greatest average accumulative reward followed by A2C and DQN. All algorithms perform well in solving the problem; however, PPO stands out of the three. Therefore, we use PPO for our result analysis in the testbed. We also tune the hyperparameters of the PPO model. These hyperparameters are essential for finding the set of hyperparameters to build the model. Without hyperparameter tuning, our model may produce sub-optimal solutions, as they fail to minimize the loss function. For hyperparameter tuning, we used Optuna [30] which is a hyperparameter optimization framework for machine learning. We tune the hyperparameters for PPO using a random sampler and medium pruner, eight parallel jobs, with 1000 trials and a maximum of 50000 steps.

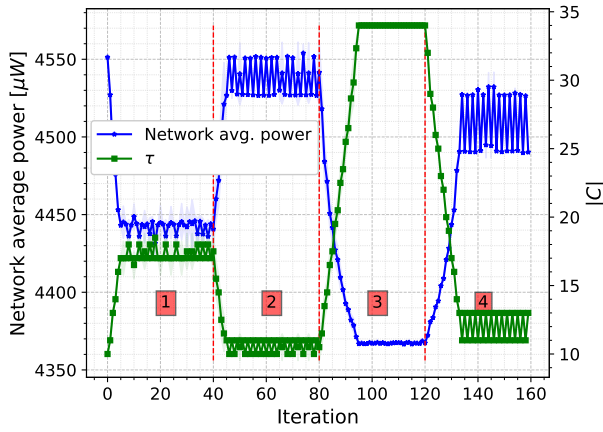
D. Experimental evaluation

We now evaluate the overall ELISE framework, including the trained agent, in the real-world testbed. We use the testbed setup previously discussed in Section V-A and

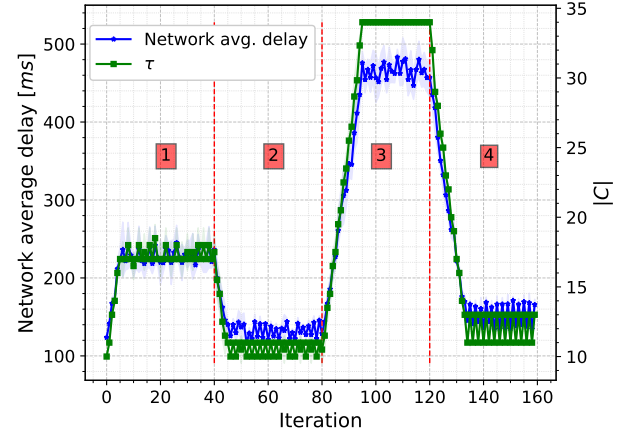
the trained agent examined in Section V-C. The agent only takes deterministic actions. We designed one single scenario that contains four distinct equally spaced user requirements: balanced, prioritized delay, prioritized power consumption, and prioritized reliability. This scenario allows us to test the agent based on the ability to select the best action given the observations and dynamic user requirements. It also enables us to observe the ability of the agent to switch between different slotframe sizes given a change in the user requirements. The evaluation consists of 10 episodes, where each episode lasts for 160 iterations. Each iteration, which is the size of the window interval, takes approximately four minutes to complete. Therefore, each episode runs for approximately 10.6 hours. The initial slotframe size ($|C|$) is set to 10 for all episodes. We plot the results individually for each performance metric against the slotframe size and the immediate reward against the slotframe size as shown in Fig 10. We use the 95% confidence interval for all charts.

1) *Balanced SDWSN*: We consider the case where the network user puts roughly the same priority on all three performance metrics. Therefore, for this specific requirement, we set values of the user coefficients α , β , and γ as 0.4, 0.3, and 0.3, respectively. The balanced requirement is applied at timestep zero and lasts for 40 timesteps (Zone 1 in Fig. 10). We can see that the trained agent needed three actions to reach the steady state value $|C|$ of 18 on average and that the RL agent balances all performance metrics equally. The network's average power consumption is roughly $4443\mu W$ (see Fig. 10a), the network's average delay is almost $220ms$ (see Fig. 10b), and the network's average reliability is around 0.95 (see Fig. 10c). The network reliability shows a larger distribution in comparison to other performance metrics. This can be attributed to the multiple sources of interference present on the testbed. Also, the relatively small size of the observations affects the reliability; therefore, missing only one packet will reduce the reliability by a few percentage points. The average immediate reward is around 1.13 (see Fig. 10d). The dispersion shown in the network reliability directly affects the distribution of the reward, but not at the same level as in the prioritized reliability case.

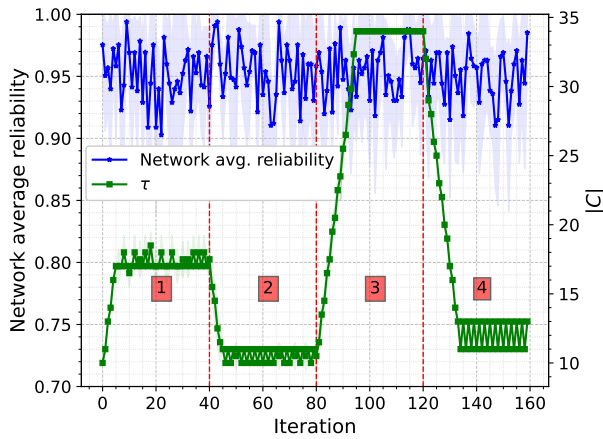
2) *Prioritized delay*: In this case, ELISE users prioritize the network delay over the network reliability and power efficiency. Therefore, we set the user coefficient values α , β , and γ as 0.1, 0.8, and 0.1, respectively. The prioritized delay requirement is applied after 40 timesteps of the start of the episode and lasts for 40 timesteps (Zone 2 in Fig. 10). It can be seen that the RL agent reacts immediately after the user requirements change. The agent self-adapts to these changes and moves the slotframe size from 18 to 11 on average in about three actions. Since the delay is prioritized, we can see that the network average delay is less than in other requirement cases. On the other hand, the network power consumption reaches the maximum across all requirement cases as the slotframe size is the lowest in the entire episode. The network average delay, power consumption, and reliability are $125ms$ (see Fig. 10b), $4540\mu W$ (see Fig. 10a), and 0.95 (see Fig. 10c), respectively. The average immediate reward is 1.13 and has a smaller distribution than in the previous case, as the contribution of



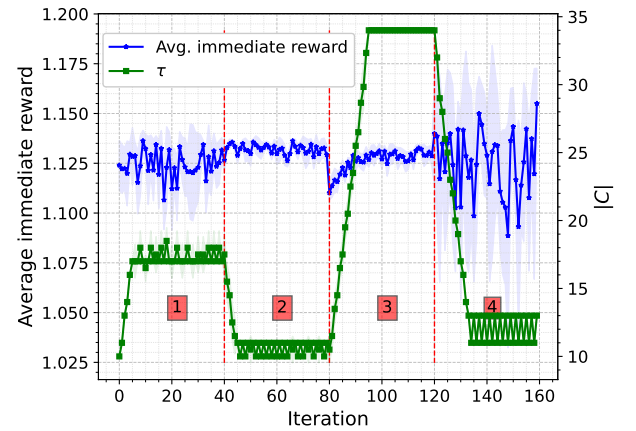
(a) Network average power consumption over the number of iterations.



(b) Network average delay over the number of iterations.



(c) Network average reliability over the number of iterations.



(d) Network average immediate reward over the number of iterations.

Fig. 10. Experimental evaluation of the ELISE framework and the RL agent for four user requirement cases. In zone 1, the balanced user requirements is running ($\alpha = 0.4$, $\beta = 0.3$, $\gamma = 0.3$). Zone 2 depicts the case for the prioritized delay user requirements ($\alpha = 0.1$, $\beta = 0.8$, $\gamma = 0.1$). Zone 3 shows the case for the prioritized power consumption user requirements ($\alpha = 0.8$, $\beta = 0.1$, $\gamma = 0.1$), and zone 4 represents the case for the prioritized reliability user requirements ($\alpha = 0.1$, $\beta = 0.1$, $\gamma = 0.8$).

the network reliability in the reward function is less as seen in Fig. 10d.

3) Prioritized power consumption: For this case, the network power efficiency is prioritized over the network delay and reliability. Thus, we set the user coefficient values for α , β , and γ as 0.8, 0.1, and 0.1, respectively. The requirements for this case are applied after 80 timesteps of the start of the episode and it also has a duration of 40 timesteps (Zone 3 in Fig. 10). The network switches from a prioritized delay network configuration to a prioritized power consumption network configuration. The RL agent detects changes in the user requirements (see the immediate reward at timestep 80) and it starts increasing the slotframe size immediately. Specifically, it increases the slotframe size from 11 up to 34 on average. It takes around 13 actions to reach the steady state value. At this point, the network experience less power consumption and higher delays overall. The network average delay, power consumption, and reliability are 470ms (see Fig. 10b), 4367μW (see Fig. 10a), and 0.95 (see Fig. 10c), respectively. The average immediate reward is 1.14 and shows less distribution than the balanced and reliable SDWSN, as the

contribution of network reliability is also less in the reward function as shown in Fig. 10d.

4) Prioritized reliability: In this situation, users prioritize network reliability over network power consumption and delay. Therefore, we selected the weights for this case as $\alpha = 0.1$, $\beta = 0.1$ and $\gamma = 0.8$. The requirements for this case are applied at timestep 120 and it lasts for 40 timesteps (Zone 4 in Fig. 10). At this time, the network switches from a prioritized power consumption configuration to a prioritized reliability network configuration. The RL agent starts decreasing the slotframe size once it detects a change in the user requirements. The RL agent takes consecutive actions to reduce the slotframe size down to 12 on average. It also takes around 13 actions to reach the steady state value. At this point, it is clear that the network experiences a low delay (145ms in Fig. 10b) and high power consumption (4510μW in Fig. 10a). The network reliability is high with a steady-state value of 0.95 on average (see Fig. 10c). Overall, the network reliability is high for all user requirement cases, but it also shows a large distribution. This can be attributed to the frequency of TSCH schedule updates. At every TSCH schedule update, the

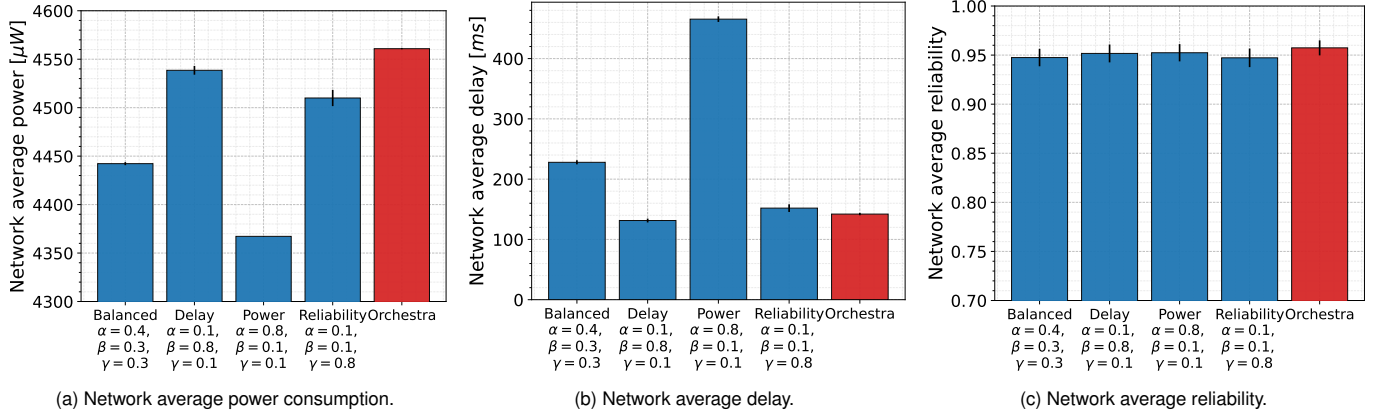


Fig. 11. Experimental comparison between the ELISE framework and Orchestra.

protocol updates its schedules, and there might be packets in the queue waiting to be transmitted. These packets are dropped due to the new schedule. To increase network reliability, users may want to increase the size of the window interval between cycles, at the expense of a less sensitive controller to changes in the environment. Moreover, the distribution of the network reliability can also be attributed to the highly dense testbed, which has multiple platforms and the ability to run multiple experiments at the same time. The distribution of the network reliability also affects the immediate reward; however, the RL agent could successfully select the correct actions despite the noise as seen in Fig. 10d.

5) ELISE and Orchestra: In Fig. 11, we present an experimental comparison between the ELISE framework and Orchestra. We consider three performance metrics as previously discussed: network average power consumption, delay, and reliability. We do not look into the immediate reward metric as Orchestra is an autonomous TSCH scheduler. This experimental evaluation considers four distinct user requirements: balanced, prioritized delay, prioritized power consumption, and prioritized reliability. The performance evaluation of network average power consumption in steady-state is shown in Fig. 11a. Orchestra has the largest average power consumption among all cases, and the prioritized power consumption case has the least average power consumption. The ELISE framework and the RL agent continuously adapt the optimal slotframe size to satisfy the user requirements, in this case, minimizing the network power consumption. Orchestra lacks this functionality forcing it to use a fixed slotframe size since the network boot. Even the prioritized delay user requirement presents less power consumption in comparison to Orchestra. This is because, in such a user requirement scenario, the power consumption weighting factor (α) still contributes to the immediate reward function. The network average delay comparison is presented in Fig. 11b. Orchestra presents a similar network average delay to that in the prioritized delay scenario, but the delay in ELISE is relatively smaller. In contrast, the prioritized power consumption requirement has the largest network delay due to the increased slotframe size to reduce power consumption. The network average reliability comparison is shown in Fig. 11c. The network reliability in

steady-state, across all user requirement cases and Orchestra, is on average 95%. Orchestra shows slightly better network reliability than ELISE. This can be attributed to the autonomous scheduling and the least number of control packets transmitted. The performance of ELISE at maximizing the network reliability for a prioritized reliability user requirement is not very notorious from the chart; however, ELISE could maintain relatively high network reliability. Also, The change in network reliability is small for relatively short slotframe sizes as shown in Fig. 8c; therefore, it is expected that network reliability will be more affected by slotframe sizes greater than 70.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose ELISE, an open-source framework that utilizes deep reinforcement learning to self-adapt network resources and optimize the slotframe size of TSCH for SDN-based IoT networks, considering dynamic user requirements. We provide a detailed description of all components involved in the framework, including the network manager module for resource orchestration, the network information and statistics module for data collection, the TSCH module for schedule processing, the routing manager module for route processing, and the ML module for hosting reinforcement learning functions.

We design a reward model based on a multi-objective function to select the optimal TSCH slotframe size that best matches the current user requirements. To expedite the training process of the reinforcement learning agent, we mathematically model the TSCH network in terms of the slotframe size. We then train and evaluate multiple state-of-the-art reinforcement learning algorithms to solve the problem.

Finally, the trained agent predicts the next valid slotframe size based on collected observations, and the framework orchestrates resources and deploys new network configurations to sensor nodes. We conduct several experiments to evaluate the performance of ELISE, considering a scenario with four distinct user requirements: balanced, prioritized delay, prioritized power consumption, and prioritized reliability. The tests assess ELISE's ability to self-adapt network resources in response to changes in user requirements. Results demonstrate

that the proposed framework can detect changes in user requirements and orchestrate network resources effectively, thereby maximizing overall network performance.

This article also highlights the complexities involved in evaluating TSCH networks using reinforcement learning in real-world deployments. The training phase is laborious and time-consuming. Although network simulators reduce the iteration time to tens of seconds, it is still insufficient to train the network within a reasonable timeframe. Thus, an approximation model of the network was employed to accelerate the training process.

During each iteration, a few unnecessary network reconfiguration packets are transmitted and lost, leading to a decrease in overall network reliability. Additionally, a few packets are dropped in the TSCH queue due to schedule updates. We plan to address this issue as a future extension of our work, and we intend to further leverage the framework to develop a reinforcement learning scheduler that can autonomously adapt to user requirements

REFERENCES

- [1] F. Deng, X. Yue, X. Fan, S. Guan, Y. Xu, and J. Chen, "Multisource energy harvesting system for a wireless sensor network node in the field environment," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 918–927, 2018.
- [2] L. M. Borges, F. J. Velez, and A. S. Lebres, "Survey on the characterization and classification of wireless sensor network applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1860–1890, 2014.
- [3] G. P. Joshi, S. Y. Nam, and S. W. Kim, "Cognitive radio wireless sensor networks: applications, challenges and research trends," *Sensors*, vol. 13, no. 9, pp. 11 196–11 228, 2013.
- [4] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on wireless communications*, vol. 1, no. 4, pp. 660–670, 2002.
- [5] S. K. Singh, P. Kumar, and J. P. Singh, "A survey on successors of LEACH protocol," *IEEE Access*, vol. 5, pp. 4298–4328, 2017.
- [6] S. M. Chowdhury and A. Hossain, "Different energy saving schemes in wireless sensor networks: A survey," *Wireless Personal Communications*, vol. 114, no. 3, pp. 2043–2062, 2020.
- [7] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-Enabled Industrial Internet (of Things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.
- [8] Y. Ha and S.-H. Chung, "Traffic-aware 6TiSCH Routing Method for IIoT Wireless Networks," *IEEE Internet of Things Journal*, 2022.
- [9] 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. 23 560–23 592, 2022.
- [10] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.
- [11] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE access*, vol. 5, pp. 1872–1899, 2017.
- [12] 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*, 2015, pp. 337–350.
- [13] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On optimal scheduling in duty-cycled industrial IoT applications using IEEE802.15.4 e TSCH," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, 2013.
- [14] T. Watteyne, M. Palattella, and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem statement," Tech. Rep., 2015.
- [15] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: Low-latency Distributed Scheduling Function for Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8688–8699, 2020.
- [16] S. Misra, S. Bera, M. P. Achuthananda, S. K. Pal, and M. S. Obaidat, "Situation-Aware Protocol Switching in Software-Defined Wireless Sensor Network Systems," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2353–2360, 2017.
- [17] L. L. Bello, A. Lombardo, S. Milardo, G. Patti, and M. Reno, "Experimental assessments and analysis of an SDN framework to integrate mobility management in industrial wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5586–5595, 2020.
- [18] T. Theodorou and L. Mamatas, "SD-MIoT: A Software-Defined Networking Solution for Mobile Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4604–4617, 2020.
- [19] T. G. Nguyen, T. V. Phan, D. T. Hoang, T. N. Nguyen, and C. So-In, "Federated Deep Reinforcement Learning for Traffic Monitoring in SDN-Based IoT Networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 4, pp. 1048–1065, 2021.
- [20] M. U. Younus, M. K. Khan, and A. R. Bhatti, "Improving the Software-Defined Wireless Sensor Networks Routing Performance Using Reinforcement Learning," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3495–3508, 2021.
- [21] H. Hajizadeh, M. Nabi, and K. Goossens, "Decentralized Configuration of TSCH-Based IoT Networks for Distinctive QoS: A Deep Reinforcement Learning Approach," *IEEE Internet of Things Journal*, 2023.
- [22] 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, p. 101089, 2022.
- [23] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with Cooja," in *Proceedings. 2006 31st IEEE conference on local computer networks*. IEEE, 2006, pp. 641–648.
- [24] M. Pilgrim and S. Willison, *Dive into python 3*. Springer, 2009, vol. 2.
- [25] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.
- [26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.
- [27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [28] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, and J. Vandaale, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 459–464.
- [29] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dornmann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021.
- [30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

F. Fernando Jurado-Lasso (GS'18-M'21) received the Ph.D. degree in Engineering and the M.Eng. degree in Telecommunications Engineering both from The University of Melbourne, Melbourne, VIC, Australia, in 2020 and 2015, respectively; a B.Eng. degree in Electronics Engineering in 2012 from the Universidad del Valle, Cali, Colombia. He is currently a postdoctoral researcher at the Embedded Systems Engineering (ESE) section of the Department of Applied Mathematics and Computer Science of the Technical University of Denmark (DTU Compute).

His research interests include networked embedded systems, software-defined wireless sensor networks, machine learning, protocols and applications for the Internet of Things.

Mohammadreza Barzegaran has been a postdoctoral research fellow in computer science at the Technical University of Denmark since 2021. His research is focused on the configuration of Fog computing platform for critical control applications. His main research interests concern Fog/Edge computing, optimization, the configuration of real-time and safety-critical systems, and co-design of control applications for real-time and safety-critical systems

J. F. Jurado received the Doctorate and MSc degree in Physics both from Universidad del Valle, Cali, Colombia, in 2000 and 1986, respectively; he also holds a BSc degree in Physics from the Universidad de Nariño, Pasto, Colombia in 1984.

He is currently a Professor with the Faculty of Engineering and Administration of the Department of Basic Science of 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 in Colombia.

Xenofon Fafoutis (S'09-M'14-SM'20) received a PhD degree in Embedded Systems Engineering from the Technical University of Denmark in 2014; an MSc degree in Computer Science from the University of Crete (Greece) in 2010; and a BSc in Informatics and Telecommunications from the University of Athens (Greece) in 2007. From 2014 to 2018, he held various researcher positions at the University of Bristol (UK), and he was a core member of SPHERE: UK's flagship Interdisciplinary Research Collaboration on Healthcare Technology. He is currently an Associate Professor with the Embedded Systems Engineering (ESE) section of the Department of Applied Mathematics and Computer Science of the Technical University of Denmark (DTU Compute). His research interests primarily lie in Wireless Embedded Systems as an enabling technology for Digital Health, Smart Cities, and the (Industrial) Internet of Things (IoT).