A Lightweight Simulation Environment for TSCH-Based Wireless Sensor Networks

A Lightweight Simulation Environment for TSCH-Based Wireless Sensor Networks

Stefano Scanzio, *Senior Member, IEEE*, Pietro Chiavassa, *Member, IEEE*, Gabriele Formis, *Student Member, IEEE*, Giacomo Paolini, *Member, IEEE*, and Gianluca Cena, *Senior Member, IEEE*

***Abstract*—Simulators are becoming increasingly important in the context of wireless networks. They allow fair comparison of different network configurations and algorithms under the same environmental conditions, and permit proper network sizing without the need for in-field experiments with real devices. Additionally, simulators enable adaptive reconfiguration of network parameters, either at runtime when employed as network digital twins or offline, by improving the training of artificial intelligence and machine learning algorithms through data augmentation. In this paper we present TSCHmodeler, a lightweight discrete-event simulator for the IEEE 802.15.4 TSCH protocol characterized by a streamlined design. Outcomes provided by the simulator were checked against experimental results acquired from a simple, real implementation, achieving a high degree of similarity, despite the simulator's limitations related to design choices favoring simplicity and usability. Then, as a concrete application of the simulator, TSCHmodeler was used to obtain latency and power consumption in a non-trivial multi-hop network topology evaluated under various configurations.**

***Index Terms*—Wireless sensor networks (WSNs), IEEE 802.15.4, discrete events simulators, industrial internet of things (IIoT), time slotted channel hopping (TSCH), ultra-low power, green networking.**

I. Introduction

INDUSTRIAL networks are heterogeneous by nature [1] and typically rely on a plurality of solutions, both wired and wireless. Among the latter, wireless sensor networks (WSNs) are playing an increasingly important role in factory/process automation [2] and for cyber-physical systems. In these contexts, high reliability and quasi-bounded transmission latency are typically required. Unlike wireless technologies like Wi-Fi and 5G, characterized by high throughput, the primary target of WSNs is to keep energy consumption as low as possible. Among the various technologies available for WSNs, a remarkable one is time slotted channel hopping (TSCH) [2], [3]. TSCH is a channel access method of IEEE 802.15.4 and helps to improve predictability by scheduling frame transmissions [4], [5]. In addition, by properly configuring its protocol parameters, it permits to finely tune the tradeoff among the three main key performance indicators (KPIs) about communication in WSNs: latency, power consumption, and reliability [6], [7].

Deployment of nodes in a WSN can be a quite a complex task [8]. The ability to evaluate network behavior since the design phase is of utmost importance to determine whether or not application constraints could be met. Network simulation is an effective method that permits estimating the above KPIs without the need to physically implement the application in the field. Besides, the time slotted nature of TSCH [9] makes modeling its behavior simpler. The main advantages of simulation, compared to a real implementation, are: cost-effectiveness, decreased times to obtain valuable results, better scalability to larger networks, and fairer comparison among different configurations. The last point depends on the fact that results obtained in the real world are unavoidably affected by the unpredictability of channel conditions, which undermines repeatability and reproducibility of experiments. Simulators can ease prototyping and evaluation of new network algorithms dramatically. Concerning TSCH, they have been used in many research activities, which include, e.g., techniques to reduce power consumption [10], [11].

A. Contribution, Advantages, and Limitations

In this work we propose a new TSCH simulator, named TSCHmodeler, which is noticeably simpler than popular solutions like TSCH-Sim [12], 6TiSCH simulator [13], OpenWSN [14], and Cooja [15]. The main advantage of TSCHmodeler, which differentiates it from other simulators and makes it a valuable alternative whenever new algorithms and techniques based on the TSCH protocol have to be defined and evaluated, is its extreme simplicity. Simulation results show an excellent match with both the experimental results on real devices and the theoretical models presented in this work, and discrepancies are practically negligible. Moreover, the use of the Python programming language eases the integration into existing industrial

Received 2 June 2025; revised 5 September 2025; accepted 5 October 2025. Date of publication 13 October 2025; date of current version 27 October 2025. This work was supported by the European Union through Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - Program "RESTART"). *(Corresponding author: Stefano Scanzio.)*

Stefano Scanzio, Pietro Chiavassa, and Gianluca Cena are with the National Research Council of Italy (CNR–IEIIT) 20133, Italy (e-mail: stefano.scanzio@cnr.it; pietrochiavassa@cnr.it; gianluca.cena@cnr.it).

Gabriele Formis is with the National Research Council of Italy (CNR–IEIIT) 10129, Italy, and also with the Politecnico di Torino, 10129 Torino, Italy (e-mail: gabriele.formis@polito.it).

Giacomo Paolini is with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna, 40136 Bologna, Italy (e-mail: giacomo.paolini4@unibo.it).

Digital Object Identifier 10.1109/TICPS.2025.3620370

applications and facilitates the configuration of new network deployments.

Regarding the limitations of our approach, the ability to quickly modify the simulator to include new features was partially achieved by limiting the implemented functionalities and, consequently, the complexity of the network model. First, TSCHmodeler focuses only on the media access control (MAC) layer, and second, the channel model it relies on is quite basic and does not explicitly involve any electromagnetic aspects (every communication link is separately described by its error probabilities on the different channels). Concerning the MAC protocol, both the TSCH matrix and the topology are configured statically (although they can be changed by the software at runtime), and only dedicated cells are supported. This type of cells is the most used in industrial contexts, because they are reserved to specific links. Relying on Python is both an advantage and a drawback. In fact, it is an interpreted language that uses dynamic typing and garbage collection, two features that negatively impact on execution speed. However, the experimental campaign we performed for evaluating execution times (reported at the end of the paper) shows that TSCHmodeler can be quietly used in large networks as well. Finally, time granularity in our simulator equals the timeslot, which means that intra-cell timings are not managed. This design choice decreases the number of events the simulator must handle, hence improving execution speed. Doing so only introduces small and practically tolerable approximations.

TSCHmodeler simplicity makes its integration in advanced application scenarios simpler. For example, it can be used by a network digital twin in the control loop for optimizing network parameters in real-time [16], [17]. The latter approach has already been proposed in the literature for other wireless protocols like Wi-Fi (see, e.g., [18]). Results provided by TSCHmodeler have been initially checked against those derived from a real network setup based on devices complying with the IETF IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) protocol suite [19], and they showed a very good match. Then, to provide a realistic use case, our simulator has been employed to obtain statistics about both latency and power consumption in a larger multi-hop network. Doing so permits to make considerations about, for instance, scheduling strategies, quickly and easily.

The following Section II of this paper describes the main characteristics of TSCH; Section III goes into the details of the TSCHmodeler simulator; Section IV shows results in terms of both validation against real devices and extension to large networks; finally, Section V draws some final remarks.

II. TSCH

Although TSCH was basically conceived as a protocol overlay for IEEE 802.15.4 [20] (the physical layer and most part of the MAC layer have been retained unchanged), its behavior differs noticeably from the original solution.

The main mechanism TSCH relies on, which defines most of its properties, is *time slotting*. A time grid is defined where the basic unit is the time slot T_{slot}, whose width is usually set to 10 or 20 ms (enough to accommodate a whole frame plus the related acknowledgment in the reverse direction). A suitable mechanism is then implemented aimed at aligning this grid on

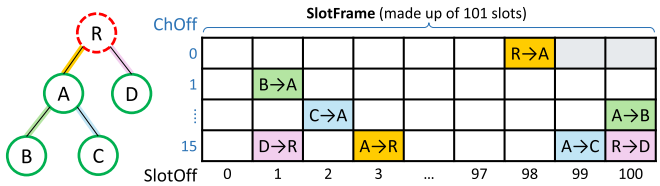

Fig. 1. Tree network topology (left) and TSCH matrix (16×101, right).

every node that takes part in a specific TSCH network. Every slot is identified by the absolute slot number (ASN), encoded as an integer on 5 B that is increased by one on every slot. In turn, the above grid is organized as a sequence of slotframes, each one made up of a fixed number of slots N_{slot} decided as part of the network configuration (typical size is $N_{\text{slot}} = 101$ slots), which repeat back-to-back indefinitely. By uniquely assigning a slot (identified by its relative position in the slotframe) to the link between a specific pair of nodes (and in a given direction), intra-network collisions can be avoided completely. Slotframe duration is $T_{\text{matr}} = T_{\text{slot}} \cdot N_{\text{slot}}$, and a typical value for it is 20 ms $\cdot$ $101 = 2.02$ s.

The second mechanism of TSCH is *channel hopping*. TSCH is able to exploit multiple channels (up to 16 can be defined in the 2.4 GHz band, numbered from 11 to 26) in order to increase the overall network throughput and improve reliability by exploiting frequency diversity. For short, a TSCH matrix is used for allocating communication resources in the time-frequency domain, whose width coincides with the slotframe and whose height equals the number of available channels. Overall, links in TSCH are allocated cells in the above matrix, each one identified by its slot and channel offsets (SlotOff and ChOff, respectively), so that multiple nodes may be transmitting at the same time without interfering with each other. A mechanism based on a pseudo-random hopping sequence is also defined, in such a way that frame transmissions on the same link performed in subsequent slotframes do not take place on the same physical channel. This technique, coupled with automatic retransmission mechanisms, permits to effectively counteract narrowband disturbance. All in all, TSCH ensures deterministic and reliable data exchanges. The only drawback is that, when disturbance becomes so high to prevent precise time-synchronization of nodes, communication is also precluded.

It is worth remembering that TSCH only takes care of frame exchanges between neighbor nodes (data-link layer). End-to-end communications across the entire WSN are accomplished by additional routing protocols (which typically belong to the network layer). For example, in 6TiSCH the Routing Protocol for Low-Power and Lossy Networks (RPL) [21] has been adopted. Since routing is not part of TSCH, operations related to the relay of packets are dealt with by TSCHmodeler in a simplified way, assuming a fixed network topology, which is satisfactory when operating in stationary conditions. All the elements that characterize the network being simulated are described schematically in Fig. 1, where links between nodes (on the left) are colored the same way as cells in the TSCH matrix (on the right). More details on TSCH operation (and in particular on channel hopping) can be found in [22].

III. Modeling TSCH

Due to its slotted nature, small TSCH networks can be easily modeled through mathematical equations. For instance, a theoretical formulation was provided in [6] for estimating latency, reliability, and power consumption. The usability of such models, however, is sometimes limited by unrealistic assumptions. As an example, in that paper network behavior is modeled satisfactorily only in the case no queuing takes place in intermediate nodes (relays) and the frame delivery ratio on links (e.g., the probability that attempts may fail) is fixed in both time and space. More fitting formulas quickly lead to intractable complexity.

Below, a description of the simulative approach followed in this work and a comparison with popular discrete-event simulators for TSCH networks are provided.

A. TSCHmodeler

TSCHmodeler is a discrete event simulator implemented in `Python` and based on the `SymPy` framework. As already stressed in the introduction, the choice to use the Python programming language was motivated by the desire to permit easy integration with other software components and to enable everybody to easily bring modifications to it. The latter point is essential to permit researchers to simulate with little effort new algorithms based on the TSCH protocol and to compare them in a proper way (i.e., under the very same environmental conditions). Consequently, TSCHmodeler focuses only on the data-link layer, ignoring both the higher layers and the physical layer. Starting from the configuration of the main network parameters and the traffic schedule (as described by the TSCH matrix), TSCHmodeler provides the KPIs relevant to industrial networks, like latency, reliability, and power consumption, in realistic application contexts. So far, the most important features implemented in the simulator include:

1) complete, static configuration of the TSCH matrix, which permits checking the effects of different schedules on performance;
2) static configuration of the topology of multi-hop networks, which permits analyzing different node placements; and,
3) separate configuration of the frame delivery probability (FDP) related to every link and channel, for both data and acknowledgment (ACK) frames, which permits transmission attempts to be modeled satisfactorily.

FDP-based error models are very simple and efficient because they do not involve aspects like signal propagation and electromagnetic disturbance, providing at the same time adequate results [22]. In this initial version, we decided to support only dedicated (non-shared) cells, because they are the preferred choice in those contexts characterized by tight requirements about latency and determinism, like distributed industrial applications.

1) Simulator Architecture: The architecture of the software is sketched in Fig. 2. As can be seen, it is very simple to understand (and, hence, to modify) and can be implemented by means of any discrete event simulator management library. The entry point of the simulator (i.e., `main` in the figure) is in charge of loading the

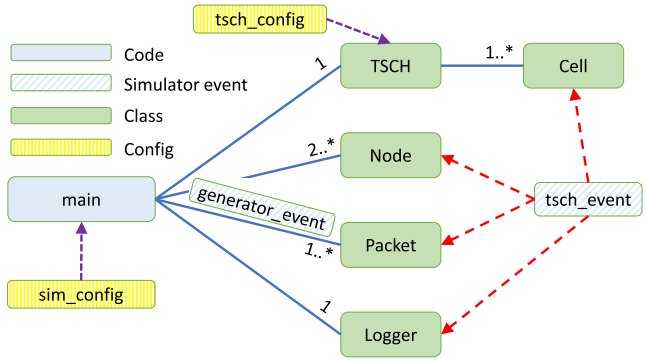

Fig. 2. High-level view of the TSCHmodeler internal architecture.

configuration files and instantiating the following classes: `TSCH`, `Node`, `Packet`, and `Logger`. There is a single instance of the `TSCH` class, which loads the configuration of the TSCH matrix from a specific configuration file (`tsch_config` in Fig. 2) and instantiates a `Cell` object for every scheduled (dedicated) cell. In particular, the simulator supports the configuration of the slot and channel offsets, the identification of the source and destination nodes of the related link (by means of integer identifiers), and the definition of a channel-dependent FDP for data and ACK frames (not necessarily they are the same, as frames have different sizes).

A second configuration file, named `sim_config`, is also read by `main`, which defines all the main configuration parameters, including the duration T_{sim} of the simulation, the number N_{slot} of slots in the slotframe (width of the TSCH matrix), the duration T_{slot} of a slot, and the maximum number N_{try} of allowed transmission attempts before a frame is discarded, plus the description of the (end-to-end) data flows that are generated by nodes and make up the traffic on the network. Concerning every data flow, TSCHmodeler permits to specify the source and destination nodes, the type of traffic (e.g., periodic), the generation period (in case of periodic traffic), and the routing information (i.e., the list of intermediate nodes that must be traversed to reach the destination).

The content of configuration files is used by TSCHmodeler to create two or more instances of the class `Node` and exactly one instance of the class `Logger`. Every instance of `Node` manages a single node (i.e., all the actions performed by the simulated device), and additionally logs statistics related to the node itself, including the overall number of transmitted and received packets, as well as information related to energy consumption. Instead, `Logger` is used to log the statistics related to every single packet along the entire path, e.g., transmission and reception times, number of transmission attempts that were actually carried out, and so on.

Since TSCHmodeler is a discrete-event simulator, specific events manage the evolution of the simulated system over time. The `generator_event` is raised by the simulator whenever the need arises to generate a new instance of class `Packet`. For periodic transmissions this event is raised cyclically, whereas for other generation laws, like the exponential one, it is raised after a random time chosen in accordance with the selected probability density function of intertimes. The simulator also

TABLE I
COMPARISON AMONG PUBLICLY AVAILABLE OPEN-SOURCE SIMULATORS FOR TSCH-BASED NETWORKS[1]

Simulator	Category	Language	Learning curve	Scalability	Protocol layers	Channel model
ns-3 [23], [24]	general-purpose	C++	high	medium	IEEE 802.15.4 6LoWPAN TSCH (third party)	complies with IEEE 802.15.4-2006
OMNet++ [25], [26]	general-purpose	C++	high	medium	IEEE 802.15.4 6TiSCH (partial, third party)	INET [27] models
OpenSim [14]	stack emulation	C (stack), Python (sim. core)	high	low	6TiSCH (full)	Pister-Hack (default)
Cooja [15]	stack emulation plus Java nodes	C (stack) Java (sim. core) Java (nodes)	high	low	6TiSCH (partial)	Unit Disk Graph (default)
6TiSCH simulator [13]	special-purpose	Python	low	high	6TiSCH (full)	Pister-Hack
TSCH-Sim [12]	special-purpose	JavaScript	low	high	6TiSCH (partial)	Unit Disk Graph Logistic Loss Pister-Hack Manual FDP config.
TSCHmodeler	special-purpose	Python	very low	high	TSCH-only	Manual FDP config.

[1]Comparison of scalability and learning curve is based on [13].

permits attaching a Python dictionary to every packet, named `ie`, which stores information related to the packet itself, such as its dimension or payload. Any traversed node can update and transform this information while relaying the packet. The `tsch_event` is the most important event from our point of view, because it manages packet forwarding to the following node (relay) and the fact that both data and ACK frames may be lost. This event is raised for every scheduled cell and is rearmed so that its period equals $T_{\text{matr}} = N_{\text{slot}} \cdot T_{\text{slot}}$, where T_{slot} is either 20 or 10 ms, depending on the implementation.

2) Power Consumption Model: Regarding power consumption, TSCHmodeler currently manages some statistics related to packet exchanges over links (mainly defined in terms of counters), including the transmission and reception of data and ACK frames, and the number of slots incurring *idle listening* (when the receiving circuitry is on but no frame is actually heard on air, causing energy to be wasted). To this extent the model described in [6] is implemented, where the energy required to send a data frame of size 127 B is $E_{\text{tx_data}} = 187\,\mu$J, to receive the related ACK frame is $E_{\text{rx_ACK}} = 79\,\mu$J, to receive a data frame of size 127 B is $E_{\text{rx_data}} = 178\,\mu$J, to send the related ACK frame is $E_{\text{tx_ACK}} = 106\,\mu$J, while the energy spent for idle listening amounts to $E_{\text{listen}} = 138\,\mu$J. The latter contribution corresponds to the energy that is spent when the receiver wakes up waiting for a frame arrival but nothing is received because the sender has no packets ready to be sent. This model is based on the OpenMote B with the TI CC2538 System-On-Chip microcontroller and the Atmel AT86RF215 radio chip.

A second power model has been additionally implemented by the simulator, which corresponds to the one in [28]. It is based on experimental measurements performed on an OpenMoteSTM device with the STM32F103RB 32-bit microcontroller and Atmel AT86RF231 radio chip. In this case, experimental measurements about energy consumption do not make any distinction between data and ACK frames, and hence only aggregate information is available for these two frames, depending on the considered side of the link: $E_{\text{tx_data}} + E_{\text{rx_ACK}} = 485.7\,\mu$J (sender), $E_{\text{rx_data}} + E_{\text{tx_ACK}} = 651.0\,\mu$J (receiver), and $E_{\text{listen}} = 303.3\,\mu$J (receiver).

It is worth noting that power consumption is highly related to the specific hardware of the node. The power model currently implemented in TSCHmodeler does not take into account frame size yet (energy consumption refers to the maximum allowed frame size, i.e., 127 B), but an extension of the simulator based on the model proposed in [29], where energy depends linearly on the message size, is already planned. At any rate, the software can be easily customized to support any kind of power model.

B. Comparison of TSCH Simulators

An overview of the available solutions for simulating TSCH networks is provided in Table I. The first kind of approach is to use general-purpose open-source network simulators such as ns-3 [23] and OMNet++ [25]. Although some implementations of the TSCH protocol are available for these simulators [24], [26], they have a steep learning curve and require time and expertise to implement and test protocol modifications, because they are written with the C++ (low-level) programming language [13].

The second kind of approach relies on the emulation of specific implementations of the mote firmware and protocol stack using the dedicated development and simulation tools of the chosen platform. This is the case of OpenSim [14] and Cooja [15]. OpenSim is a Python simulator for OpenWSN, which is a pure-C implementation of the full 6TiSCH protocol stack (including TSCH, 6LoWPAN, RPL, CoAP, etc.). The emulated code coincides with the 6TiSCH implementation and applications executed by the mote. When OpenWSN is compiled on a conventional PC, OpenSim simulates the network for interconnecting motes. Instead, Cooja [15] is a Java simulator for the Contiki(-NG) operating system for sensor nodes [30]. Contiki(-NG) implements the minimal configuration of the 6TiSCH stack [31], [32]. The entire software stack can be emulated either at the machine-instruction level or by compiling it for a conventional PC. Non-Contiki nodes, developed in Java, can also be added to the simulation. For both simulators, when emulation is used, complexity is exactly the same as the code running in the real mote, which makes implementation and experimentation of new algorithms impractical.

The final kind of approach consists in using TSCH-specific network simulators. This is typically recommended, as they are simpler than previous solutions and contain more up-to-date protocol features. TSCH-Sim [12], [33], [34] is a JavaScript-based simulator designed to improve the scalability and extensibility of 6TiSCH stack simulations. The simulator is divided into three layers: interface, network, and device layers. The interface layer handles global configuration, routing, and logging, including power consumption models based on hardware characteristics. The network layer implements various propagation models and allows separate configurable transmission probabilities for every channel. At the device layer, TSCH-Sim supports many routing protocols (e.g., RPL) and schedulers. In addition, TSCH-Sim can model node mobility. Due to the large number of features implemented by the simulator (it covers 6TiSCH and not only TSCH) and the JavaScript programming language it relies on, its complexity is much higher than TSCHmodeler.

Of the analyzed simulators, 6TiSCH simulator [13] is the one that most closely resembles TSCHmodeler. It is written in Python and simulates the whole 6TiSCH stack, only considering layers from the MAC up. Messages are not byte-accurate but only convey semantic-relevant parameters. The propagation model is the same as OpenSim (Pister-Hack) and received signal strength indicator (RSSI) values are converted to transmission success probabilities. An energy consumption model is defined by considering the different slot types (idle, transmission, reception, etc.) and the kinds of operations carried out inside them (e.g., radio TX/RX, CPU, etc.). Application traffic can be modeled according to different probability distributions, and traffic bursts can be scheduled. As for TSCHmodeler, the 6TiSCH simulator allows new users to quickly deploy simulations, avoiding the steep learning curve required by other platforms. There are, however, consistent differences between them. In fact, TSCHmodeler only simulates the TSCH protocol, and consequently it is much simpler than the 6TiSCH simulator, allowing users (typically, researchers) to concentrate only on the IEEE 802.15.4 MAC layer. In particular, TSCHmodeler permits a quick customization of the simulator's features and enables fast implementations of new experiments, algorithms, and digital models of the environment. Concerning scalability, when considering hierarchical networks we observed a linear growth in execution time with the number of nodes. This is similar to what is achieved by TSCH-Sim, which in turn provides optimizations over the 6TiSCH simulator.

IV. Results

Three experimental campaigns were carried out aimed at: a) assessing the similarity between real and simulated results, b) showing how TSCHmodeler can be used to obtain statistics about traffic and power consumption in realistic network configurations, and, c) analyzing the scalability of the proposed simulator for large networks with a sizable number of nodes. The parameters used for simulations, reported in Table II, have been either derived from the actual configuration of real devices or obtained from experiments carried out on testbeds including such real devices.

TABLE II
Simulation Parameters (Protocol/Channel/Power Model)

Quantity	Description	Typical value
N_{slot}	Slots in a slotframe	101
T_{slot}	Slot duration	20 ms
N_{try}	Maximum allowed tx attempts	16
T_{matr}	Duration of a slotframe	2.02 s
ϵ	Frame error probability	0.0413
$E_{\text{tx\_data}}$	Energy to transmit a data frame	187 μJ
$E_{\text{tx\_ACK}}$	Energy to transmit an ACK frame	106 μJ
$E_{\text{rx\_data}}$	Energy to receive a data frame	178 μJ
$E_{\text{rx\_ACK}}$	Energy to receive an ACK frame	79 μJ
E_{listen}	Energy wasted for idle listening	138 μJ

TABLE III
Comparison Between Statistics on Latency Measured on Real Devices and Those Simulated Using TSCHmodeler (Simple)

Type	Duration	$d_{\min}$	μ_d	σ_d (ms)	d_{99}	$d_{\max}$	Eval. Time
Measured (real)	1week	960	2125.3	817.4	4753	8059	1 week
Simulated	1 week	940	2112.9	856.9	4840	12980	2.3 s
Simulated	20 years	940	2113.8	840.6	4840	12980	2304 s

A. Validation

This first campaign aims to determine how much experimental and simulated results differ from the point of view of latency. Results are also checked against known theoretical formulas, to prove their correctness. To this purpose, a real TSCH network was setup, which is composed of two OpenMote B devices running the 6TiSCH protocol stack implemented by the OpenWSN operating system. The devices are exactly the same as those employed in [6] for obtaining the power consumption model used in this work.

The `ping` command was used to repeatedly generate an internet control message protocol (ICMP) Echo request of size 127 B every 2 minutes, which is directed from the root device to the (only) leaf node. On its reception the leaf node reacts by sending an ICMP Echo reply back to the root, enabling round-trip time (RTT) measurement. The overall duration of the experiment was one week. The architecture of the testbed we employed (including the manual configuration of the TSCH matrix) is sketched in Fig. 3(a) under the name *simple topology*. Due to the time slotted nature of the TSCH protocol, evaluating communication on a single link between two nodes is enough for validating also multi-hop topologies. Despite this statement does not account for queuing inside nodes, it must be remembered that this phenomenon concerns computing (and not communication on air), and affects in practically the same way both real nodes and the simulated ones. These aspects have been properly debugged and analyzed when developing the simulator.

Statistics about latency (RTT) are reported in the first row of Table III. They are obtained from the log produced by the ping command by means of the tool `TSCHStats.py`, downloadable from [35]. In particular, the table reports the minimum ($d_{\min}$), average (μ_d), standard deviation (σ_d), 99-percentile (d_{99}), and maximum ($d_{\max}$) of the latency.

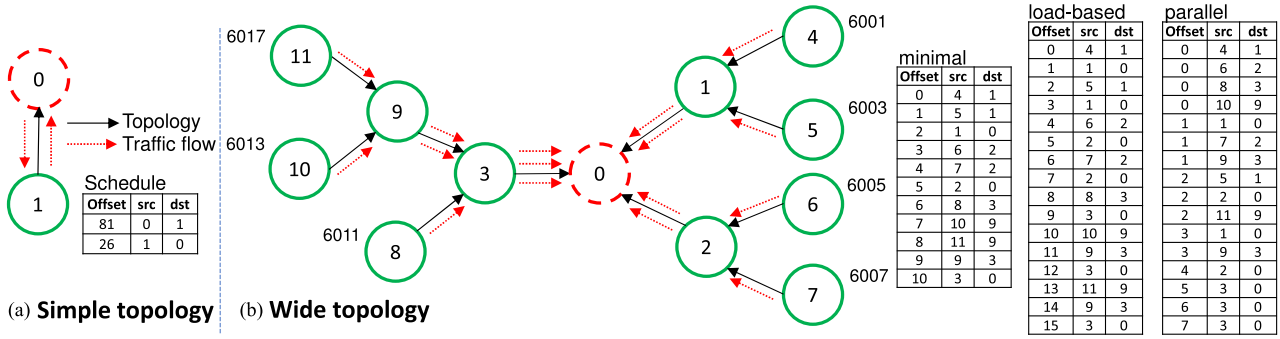

Fig. 3. The two network topologies analyzed in the paper: comparison with the experimental setup (simple, on the left) and evaluation of a non-trivial realistic scenario (wide, on the right). Numbers near leaf nodes represent the related generation period expressed in time slots.

In the absence of transmission errors, latency only depends on the configuration of the TSCH matrix. The tools provided by the OpenWSN operating system permit to know which slots in the TSCH matrix are scheduled for communication between any two nodes (they are selected by 6TiSCH automatically). In the context of this experimental campaign, slot 81 was scheduled for communication between node 0 (the root) and node 1 (the leaf), while slot 26 was reserved for communication in the opposite direction (link $1 \rightarrow 0$). The two transmission opportunities (cells) involved with the ping request ($0 \rightarrow 1$) and reply ($1 \rightarrow 0$) are displaced by $101 - 81 + 26 = 46$ slots. Since $T_{\text{slot}} = 20\,\text{ms}$, the theoretical minimum latency $d_{\text{min}}^{\text{th}}$ is about $(46+1) \cdot T_{\text{slot}} = 940\,\text{ms}$. The value we measured for d_{min} was instead 960 ms. The small difference is due to the fact that real devices take some time to perform operations like queuing a packet for transmission and passing a received packet up to the application layer.

The TSCH matrix of TSCHmodeler was configured in exactly the same way as real devices, and the same holds for the other network parameters (see Table II). In particular, the frame error probability $\epsilon = 0.0413$ was set equal to the measured frame loss ratio obtained from the experimental testbed. Doing so is essential to permit the simulator to faithfully reproduce the behavior of the real link, as actual channel conditions are likely to vary abruptly and unpredictably. Starting from ϵ, the theoretical value of the average latency[1] can be computed as

$$\mu_d^{\text{th}} = d_{\text{min}}^{\text{th}} + \left(\frac{1}{2} + N_{\text{hop}} \cdot \left(\frac{1}{1-\epsilon} - \frac{N_{\text{try}} \cdot \epsilon^{N_{\text{try}}}}{1-\epsilon^{N_{\text{try}}}} - 1 \right) \right) \cdot T_{\text{matr}}, \quad (1)$$

where $N_{\text{hop}} = 2$ to account for the ICMP request and reply, and is equal to 2124.0 ms, which is practically the same as the measured value (2125.3 ms). The difference between the measured and simulated values of the average latency ($\sim 10\,\text{ms}$) can be explained by remembering that traffic in the simulator is generated synchronously with the beginning of the slot, whereas in the testbed the packet generation process (`ping` command) and the grid of slots are not aligned, which implies that the queuing time for a frame also includes a random contribution uniformly distributed between zero and the slot time (half, on average). This effect is instead considered in Eq. (1).

[1]Equation (1) was obtained from (16) and (18) described in [6].

The second row of Table III reports statistics obtained from simulation over a one-week timeframe. As can be seen, they closely resemble those observed on real devices. The only noticeable exception concerns d_{max}, that for the simulation was about 5 s larger than in real experiment. This behavior is unsurprising: in fact, the considered timeframe (one week) is quite short, which means that there is a non-negligible probability that events leading to worst cases (transmission failures) differed tangibly in the simulated and real environments.

To test the simulation speed of TSCHmodeler, the same experiment was repeated with a duration of 20 years, and results are summarized in the third row of the table. Execution times are reported in the rightmost column. As can be seen, the 20-year simulation only took 2304 s (less than 40 minutes).

B. Simulating Realistic Networks

After verifying the consistency of the results provided by the simulator with what is observed in a real network, TSCHmodeler was exploited to analyze the *wide topology* depicted on the right side of Fig. 3. In application scenarios characterized by a large number of nodes (tens, and even hundreds), simulation is always valuable (and sometimes essential) because it allows very complex architectures to be analyzed quickly and inexpensively. As said before, simulators permit different network algorithms to be compared fairly, since spectrum conditions are the same for all the execution instances. Hence, they assist designers in correctly configuring the network parameters to meet the application requirements (in terms of, e.g., latency and energy consumption).

The *wide topology* presented in this section mimics a multi-hop network where some leaf nodes are located two hops away from the root while others (i.e., nodes 10 and 11) are located three hops away. Every leaf node generates a periodic flow whose period is about 2 minutes, which is typical of sensing applications aimed at acquiring physical quantities with slow dynamics, e.g., for environmental monitoring. In application contexts where faster sampling rates are required, other protocols with higher throughput and possibly real-time guarantees are used (e.g., Wi-Fi or 5G), which are not compatible with ultra-low power requirements. Generation periods were selected

as coprime numbers to prevent the traffic flows from being synchronized. In this way, offsets between packets sent by different nodes are not fixed. On the one hand, the selected periods for packet flows satisfactorily model real devices and, in particular, the tolerances of their quartz oscillators. On the other hand, they permit the simulation to consider all the possible packet queuing orders (or better, a large and statistically relevant number of them) in the intermediate nodes of the network.

This simulation campaign aims to show the use of TSCHmodeler to quickly compare the effect of different configurations for the TSCH matrix (which defines the schedule of transmission opportunities between network nodes) on latency and power consumption. For instance, referring to the network topology in Fig. 3(b), if the transmission opportunity $9 \rightarrow 3$ is moved before $11 \rightarrow 9$ in the schedule, a substantial increase is experienced in the transmission latency. In fact, after the packet is transmitted from node 11 to node 9, it has to wait for a time equal to $T_{\text{matr}} - T_{\text{slot}} = 2$ s before being forwarded from node 9 to node 3. In this work, four schedules are analyzed, namely, *minimal*, *load-based*, *parallel*, and *load-parallel-redundant*. The content of the TSCH matrices (i.e., the schedules) for the first three configurations are reported in the rightmost part of Fig. 3. For the sake of simplicity, in the experimental campaign all links were configured to have the same frame error probability.

In the *minimal* configuration, a single cell is allocated to each link. It is important to note that, in this configuration, if a transmission attempt fails its retransmission occurs in the next slotframe (after $T_{\text{matr}} = 2.02$ s), which leads to a tangible increase in latency. The *load-based* configuration allocates to any given link a number of cells equal to the number of flows crossing that link (red arrows). For instance, the link $9 \rightarrow 3$ is used by two flows (from node 10 to node 0 and from node 11 to node 0), and consequently two transmission opportunities are scheduled for it in the TSCH matrix (see slot offsets 11 and 14 in the TSCH matrix named *load-based* in Fig. 3(b)).

The *parallel* configuration is similar, but the same slot offset in the schedule is used to accommodate more than one cell at the same time exploiting channel offsets. Clearly, concurrent transmissions can be performed only on links involving completely different pairs of nodes. Finally, the *load-parallel-redundant* (*LPR*) configuration extends the previous two configurations (*load-based* and *parallel*) by providing even more redundancy (the number of cells scheduled for any given link is doubled). In terms of slot offset, the second cell reserved for every specific pair of nodes was located immediately after the first transmission opportunity. The rationale behind this schedule is to enable an immediate retransmission when the first attempt fails, so as to lower transmission latency.

For space reasons, the configuration of the TSCH matrix for the LPR case was not included in this work, but in a specific repository [36][2] together with the logs obtained by the simulator, from which statistics were derived. In addition, it contains the source code of the TSCHmodeler implementation we used in this work.

[2]The dataset and the current version of TSCHmodeler (file `TSCHmodeler.zip`) can be downloaded from the following Zenodo repository [36], which also includes the schedule of all the configurations analyzed in this work.

TABLE IV
STATISTICS ON LATENCY AND POWER CONSUMPTION (NETWORK/NODE) OBTAINED WITH TSCHMODELER (WIDE)

	Node	$d_{\min}$	μ_d	σ_d	d_{99}	$d_{\max}$	P	P_{listen}
		(ms)					(μW)	
Minimal	All	40	1290.2	908.8	4060	16000	808.1	732.3
	8	100	1277.6	845.2	4000	12140	2.308	0.0
	11	60	1398.8	1031.8	4920	16000	2.306	0.0
Load-b.	All	40	1183.3	786.1	3820	12120	1149.7	1073.9
	8	40	1129.7	723.6	3640	12120	2.308	0.0
	11	60	1306.3	898.7	3940	11940	2.306	0.0
Parallel	All	40	1186.2	759.2	3800	12040	1149.7	1073.9
	8	120	1207.9	723.6	3720	12040	2.308	0.0
	11	80	1254.0	830.3	3940	11300	2.306	0.0
LPR	All	40	1060.0	590.4	2060	7240	2242.8	2167.0
	8	80	1086.8	589.2	2080	6020	2.308	0.0
	11	60	1072.7	594.9	2060	7240	2.306	0.0

Results for a 20-year simulation are reported in Table IV. The first row of each configuration concerns statistics evaluated on all nodes in the network, while statistics specifically related to nodes 8 (two hops) and 11 (three hops) are included in the second and third rows of any configuration. Besides statistics related to the transmission latency, the last two columns of any row report the power consumption P (for all nodes and separately for nodes 8 and 11) and the contribution P_{listen}, which is related to the idle listening phenomenon.

As can be seen, the average latency of the *wide* topology is always shorter than for the *simple* topology. This is due to the specific schedules we used in that case, which aim to minimize latency by a proper selection of the order and position of the scheduled cells. As expected, the closer the node to the root, the shorter the latency (see nodes 8 and 11). Regarding power consumption, results for nodes 8 and 11 are similar because both are leaves. This implies that they access the network only for transmitting locally generated packets, but no cells are allocated for frame reception. Hence, they do not suffer from idle listening and $P_{\text{listen}} = 0.0\,\mu$W.

For the *minimal* configuration, the power consumed by relay nodes 9 (two hops to the root), 3 (one hop to the root), and 0 (network root) is 143.7 μW, 147.2 μW, and 213.6 μW, respectively (not shown in the table), noticeably higher than leaves. As results clearly highlight, the closer the considered node to the root, the higher energy consumption, because traffic generated by the leaves in a WSN typically converges to the gateway and is merged by relay nodes along upward paths (a similar split effect is observed in the downward direction). Hence, the number of flows crossing every intermediate node becomes progressively higher, and the same holds for the traffic they have to relay (packet rate). In addition, relays do suffer from idle listening.

The *load-based* configuration greatly improves all the statistics related to communication latency, because having two scheduled cells for links $1 \rightarrow 0$, $2 \rightarrow 0$, and $9 \rightarrow 3$, and three scheduled cells for the link $3 \rightarrow 0$, ensures that, in the absence of

link failures (no retransmissions), every packet can be transferred from the source to the destination node in the same slotframe. As a downside, there is a substantial increase in power consumption P, which rises from 808.1 μW to 1149.7 μW because of idle listening.

Power consumption does not vary for the *parallel* configuration, since the number of scheduled cells (15, in the example) remains the same as the *load-based* configuration. Rather counterintuitively, latency statistics are slightly worse. This is due to the fact that some transmissions are performed earlier compared to the *load-based* case, because they are carried out in parallel with other links. For example, the transmission scheduled for link $6 \rightarrow 2$ is performed in the cell with slot offset 0, but the packet remains queued for a longer time inside node 2, before being relayed to node 0 in the cell with slot offset 2. By contrast, in the *load-based* configuration, a packet arriving in slot 4 on link $6 \rightarrow 2$ can be relayed immediately on link $2 \rightarrow 0$ in slot 5.

The last *LPR* configuration shows a noticeable decrease in communication latency. This is due to the higher number of transmission opportunities between every pair of nodes (they are doubled). The effect on latency is more evident in real-time statistical indicators such as $d_{\mathrm{p}_{99}}$ and $d_{\max}$. In fact, if the first transmission attempt fails, retransmission takes place in the immediately following slot. Without this over-provisioning of transmission opportunities, the sender must wait for a time equal to $T_{\mathrm{matr}} = 2.02\,\mathrm{s}$ before retrying the transmission, increasing percentiles and maximum latency noticeably.

At the same time, there is a substantial increase in power consumption for the whole network, which grows up to 2242.8 μW (from 808.1 μW for the *minimal* configuration and 1149.7 μW for the *load-based* and *parallel* configurations). This is due to the additional transmission opportunities reserved in the *LPR* schedule, which in TSCH networks lead to energy waste because of idle listening, even if they are not actually used.

The above kind of analysis, when performed using a fast and lightweight tool like TSCHmodeler, can be exploited by a network digital twin to determine the effect of changes to network parameters before they are applied to real devices, both in the initial network configuration phase and for reconfiguration at runtime. Given the FDP for transmissions on air (that can be estimated dynamically by network nodes), the simulator can be used, e.g., to predict how different network/application configurations affect performance indicators about communication that are relevant to industrial systems (latency, reliability, power consumption, etc.). As already stressed, TSCH modeler simplicity makes its integration and customization easier, in accordance with the application's needs.

C. Scalability

To assess TSCHmodeler scalability versus the network size, the time taken for simulating two networks composed of 40 and 121 nodes for one year was evaluated. These configurations, we named *40-node* and *121-node*, respectively, have a tree topology and all nodes have degree three (i.e., each node, excluding leaves, has exactly 3 children). The height for the *40-node* topology is 4 (three layers below the root), while for *121-node* it is 5.

TABLE V
ANALYSIS OF SCALABILITY AND EXECUTION TIMES OF TSCHMODELER WITH DIFFERENT CONFIGURATIONS (ONE WHOLE YEAR IS SIMULATED)

Cond.	Ev. ($\cdot 10^6$)	Redirect			No print		
		Time (s)	Ev./s	Speedup	Time (s)	Ev./s	Speedup
Simple	31.5	77	409140	409558×	41	772355	769171×
Minimal	174	439	395153	71836×	239	727300	131950×
Load-b.	252	622	404269	50701×	333	754745	94703×
Parallel	252	638	394165	49430×	356	707342	88584×
LPR	501	1345	372704	23447×	693	723757	45507×
40-node	616	1873	328725	16837×	928	663557	33983×
121-node	1894	5579	316892	5653×	2920	648965	10800×

Each one of the leaf nodes (i.e., 27 and 81 nodes for the two configurations, respectively) generates a cyclic flow with a period of 2 minutes directed toward the root node. The slotframe schedule was obtained by randomly selecting the cells used for every individual link in the tree. The two sample configurations we analyzed in this experimental campaign represent a medium-to-large and a large network, comparable to those that can be reasonably found in typical industrial settings. A common usage of WSNs in such scenarios is environmental sensing (e.g., acquisition of temperature and humidity to check storage conditions of food products). They can be also exploited to perform actuation, in those cases where timing requirements are (much) less stringent than the typical operating conditions supported by Industrial Ethernet, Wi-Fi, and 5G.

Simulation was conducted on a Linux machine equipped with an Intel Xeon w7-3455 Processor running at 4.8 GHz with a Python interpreter version 3.10.12.

Table V summarizes the main results of the analysis for every single configuration considered in this work (including the larger ones introduced above), after shortening the duration of simulations to one year. The first column of the table is the overall number of events managed by the simulator (it counts the number of *simulation_event* and *tsch_event*). As expected, they increase proportionally to the complexity of the network. The other results refer to two specific settings: *Redirect*, where the output of the simulator is redirected into a file, and *No print*, where all the print instructions are commented out. For both settings, the execution time, the number of events per second, and the speedup factor are reported. Speedup represents how many times the simulator is faster than real time. As can be seen by results in the table, about half of the execution time is taken by print instructions. Even in the most challenging *121-node* configuration, the execution time when nothing is printed is limited to 2920 s (i.e., $\sim$ 49 minutes), which is 10800 times faster than real-time operations in the physical world.

Regarding memory usage, the resident set size (RSS), which represents the amount of physical memory used by the simulator, is 13.88 MB for the *121-node* configuration, while the virtual memory size (VMS) amounts to 24.22 MB. Interestingly, memory usage is practically the same for all the considered configurations, since it mainly depends on the Python libraries we used. For example, the *40-node* configuration has the same RSS while VMS is equal to 24.15 MB.

Above results show that, despite relying on a pure-Python implementation, TSCHmodeler performance is fully adequate for dealing with real networks and is fast enough to be included in network digital twins.

V. Conclusion

Network simulators are powerful tools, which can directly impact the real world by easing equipment (re-)configuration, even at runtime. This includes those cases where the simulator is part of a network digital twin, included in a sensing/parameterization loop, or is used for suitably augmenting training data to improve the performance of artificial intelligence and machine learning models.

The TSCHmodeler lightweight simulation environment presented in this work is a discrete event simulator for IEEE 802.15.4 TSCH-based networks that, due to its extreme simplicity and ease of use, can be easily employed in above contexts to optimize performance and increase the system's intelligence. Besides, it can be adopted to quickly perform preliminary evaluations, before moving to experimentation on real setups.

Experimental results about communication latency show that not only the simulator provides a very good match with simple real networks equipped with commercial devices, but it can be profitably used for analyzing the behavior of more complex networks, providing useful results with little effort.

Future steps include the implementation of a larger set of features for the simulator, some use cases where it is employed to analyze more complex network configurations, and finally the release of TSCHmodeler as open-source software.

Acknowledgment

The authors wish to thank Dr. Mohammad Ghazi Vakili, who participated in the early stages of this research activity.

References

[1] S. Scanzio, L. Wisniewski, and P. Gaj, "Heterogeneous and dependable networks in industry - A survey," *Comput. Ind.*, vol. 125, 2021, Art. no. 103388.

[2] A. Tabouche, B. Djamaa, and M. R. Senouci, "Traffic-aware reliable scheduling in TSCH networks for industry 4.0: A systematic mapping review," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 4, pp. 2834–2861, Fourthquarter 2023.

[3] A. Eloudrhiri Hassani, A. Sahel, A. Badri, and I. El Mourabit, "Multiple channel access techniques in industrial IoT: Evaluation case of time-slotted channel hopping," *Wireless Netw.*, vol. 30, no. 5, pp. 3119–3133, 2024.

[4] Y. H. Pratama, S.-H. Chung, and D. Z. Fawwaz, "Low-latency and Q-learning-Based distributed scheduling function for dynamic 6TiSCH networks," *IEEE Access*, vol. 12, pp. 49694–49707, 2024.

[5] A. Kalita and M. Gurusamy, "On-the-fly autonomous slot allocation in 6TiSCH-Based industrial IoT networks," *IEEE Trans. Ind. Informat.*, vol. 20, no. 7, pp. 9365–9374, Jul. 2024.

[6] 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.

[7] G. Cena, C. G. Demartini, M. G. Vakili, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluating and modeling IEEE 802.15.4 TSCH resilience against Wi-Fi interference in new-generation highly-dependable wireless sensor networks," *Ad Hoc Netw.*, vol. 106, 2020, Art. no. 102199.

[8] R. Priyadarshi, B. Gupta, and A. Anurag, "Deployment techniques in wireless sensor networks: A survey, classification, challenges, and future research issues," *J. Supercomputing*, vol. 76, no. 9, pp. 7333–7373, 2020.

[9] M. Nabi, M. Habibollahi, and H. Saidi, "Time hopping: An efficient technique for reliable coexistence of TSCH-Based IoT networks," *IEEE Internet Things J.*, vol. 10, no. 15, pp. 13837–13848, Aug. 2023.

[10] S. Scanzio, G. Cena, and A. Valenzano, "Enhanced energy-saving mechanisms in TSCH networks for the IIoT: The PRIL approach," *IEEE Trans. Ind. Informat.*, vol. 19, no. 6, pp. 7445–7455, Jun. 2023.

[11] G. Cena, S. Scanzio, and A. Valenzano, "Ultra-Low Power Wireless Sensor Networks Based on Time Slotted Channel Hopping with Probabilistic Blacklisting," *Electron.*, vol. 11, no. 3, 2022, Art. no. 304.

[12] A. Elsts, "TSCH-Sim: Scaling up simulations of TSCH and 6TiSCH networks," *Sensors*, vol. 20, no. 19, 2020, Art. no. 5663.

[13] E. Municio et al., "Simulating 6TiSCH networks," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 3, 2019, Art. no. e3494.

[14] T. Watteyne et al., "OpenWSN: A standards-based low-power wireless development environment," *Trans. Emerg. Telecommun. Technol.*, vol. 23, no. 5, pp. 480–493, 2012.

[15] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. 2006 31st IEEE Conf. Local Comput. Netw.*, 2006, pp. 641–648.

[16] L. U. Khan, Z. Han, W. Saad, E. Hossain, M. Guizani, and C. S. Hong, "Digital twin of wireless systems: Overview, taxonomy, challenges, and opportunities," *IEEE Commun. Surveys Tut.*, vol. 24, no. 4, pp. 2230–2254, Fourthquarter 2022.

[17] N. P. Kuruvatti, M. A. Habibi, S. Partani, B. Han, A. Fellan, and H. D. Schotten, "Empowering 6G communication systems with digital twin technology: A comprehensive survey," *IEEE Access*, vol. 10, pp. 112158–112186, 2022.

[18] S. Scanzio et al., "Multi-link operation and wireless digital twin to support enhanced roaming in Next-Gen Wi-Fi," in *Proc. IEEE 20th Int. Conf. Factory Commun. Syst.*, 2024, pp. 1–4.

[19] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Commun. Surveys Tut.*, vol. 22, no. 1, pp. 595–615, Firstquarter 2020.

[20] IEEE, *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4-2015 (Revision of IEEE Standard 802.15.4-2011), 2016.

[21] R. Alexander et al., "RPL: IPv6 routing protocol for low-power and lossy networks," RFC 6550, 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6550

[22] G. Cena, S. Scanzio, M. G. Vakili, C. G. Demartini, and A. Valenzano, "Assessing the effectiveness of channel hopping IEEE 802.15.4 TSCH networks," *IEEE Open J. Ind. Electron. Soc.*, vol. 4, pp. 214–229, 2023.

[23] NSNAM, "NS-3," (n.d.), 2025. [Online]. Available: https://www.nsnam.org/

[24] "NS-3-TSCH," (n.d.), 2025. [Online]. Available: https://github.com/GmelaN/lr-wpan-TSCH

[25] "OMNeT discrete event simulator," (n.d.), 2025. [Online]. Available: https://omnetpp.org/

[26] Y. Shudrenko, D. Plöger, K. Kuladinithi, and A. Timm-Giel, "A novel approach to enhance the end-to-end quality of service for avionic wireless sensor networks," *ACM Trans. Internet Technol.*, vol. 22, no. 4, pp. 95:1–95:29, 2022.

[27] "INET Framework," (n.d.), 2025. [Online]. Available: https://inet.omnetpp.org/

[28] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. J. Pister, "A realistic energy consumption model for TSCH networks," *IEEE Sensors J.*, vol. 14, no. 2, pp. 482–489, Feb. 2014.

[29] G. Cena, S. Scanzio, and A. Valenzano, "Enabling listening suspension in the time slotted channel hopping protocol," in *Proc. IEEE 17th Int. Conf. Factory Commun. Syst.*, 2021, pp. 19–26.

[30] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, 2004, pp. 455–462.

[31] "TSCH and 6TiSCH — Contiki-NG documentation," (n.d.), 2025. [Online]. Available: https://docs.contiki-ng.org/en/develop/doc/programming/TSCH-and-6TiSCH.html

[32] S. Duquennoy, A. Elsts, B. A. Nahas, and G. Oikonomo, "TSCH and 6TiSCH for contiki: Challenges, design and evaluation," in *Proc. 13th Int. Conf. Distrib. Comput. Sensor Syst.*, 2017, pp. 11–18.

[33] T. Ara, T. Singh, A. Vatankhah, and R. Liscano, "Enhancement of the TSCH-SIM simulator to support manual scheduling and routing," *Procedia Comput. Sci.*, vol. 203, pp. 61–68, 2022.

[34] T. Ara, A. Vatankhah, and R. Liscano, "Enhancement of the TSCH-Sim simulator via web service interface to support co-simulation optimization," *J. Ubiquitous Syst. Pervasive Netw.*, vol. 18, pp. 69–76, 2023.
[35] S. Scanzio et al., "Wireless sensor networks dataset (TSCH a compromise between reliability, power consumption, and latency)," *IEEE Dataport*, Nov. 2020, doi: 10.21227/fg62-bp39.
[36] S. Scanzio, P. Chiavassa, and G. Cena, "Simulated TSCH dataset using different slotframe matrix configurations," *Zenodo* Sep. 2025, doi: 10.5281/zenodo.17024133.

Stefano Scanzio (Senior Member, IEEE) received the Laurea and Ph.D. degrees in computer science from Politecnico di Torino, Turin, Italy, in 2004 and 2008, respectively. From 2004 to 2009, he was with the Department of Computer Engineering, Politecnico di Torino, Torino, Italy, where he was involved in research on speech recognition and classification methods and algorithms. Since 2009, he has been with the National Research Council of Italy, where he is currently a Senior Researcher with the Institute CNR-IEIIT. He teaches several courses on computer science with Politecnico di Torino and Università degli Studi di Pavia. He has authored or coauthored more than 120 papers in international journals and conferences, in his research interests which include industrial communication systems, real-time networks, wireless networks, and artificial intelligence. He took part in the program and organizing committees of many international conferences of primary importance in his research areas. He was the recipient of the the 2017 Best Paper Award from IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the Best Paper awards for three papers presented at the IEEE Workshops on Factory Communication Systems, in 2010, 2017, and 2019, and for a paper presented at the IEEE International Conference on Factory Communication Systems, in 2020. He is an Associate Editor for IEEE ACCESS, *Ad Hoc Networks* (Elsevier), and *Electronics* (MDPI).

Pietro Chiavassa (Member, IEEE) received the Ph.D. degree in computer and control engineering from Politecnico di Torino, in 2024. He is currently a Postdoc Researcher with the National Research Council of Italy, within the CNR-IEIIT Institute. His research interests include IoT, wireless sensor networks, Wi-Fi, security and privacy, and quantum computing.

Gabriele Formis (Student Member, IEEE) received the B.Sc. degree in mechanical engineering and the M.Sc. degree in automation and control engineering from the Politecnico di Milano, Milan, Italy, in 2018 and 2020, respectively. He is currently a Research Associate with the Institute of Electronics, Computer and Telecommunication Engineering, National Research Council of Italy (CNR-IEIIT), Italy. His research interests include artificial intelligence, wireless networks, and autonomous driving.

Giacomo Paolini (Member, IEEE) received the M.Sc. degree in biomedical engineering and the Ph.D. degree in electronics, telecommunications, and information technologies engineering from the University of Bologna, Bologna, Italy, in 2016 and 2021, respectively. He joined the Interdepartmental Center for Industrial Information and Communication Technologies Research (CIRI ICT), University of Bologna, as a Research Fellow within the EU-supported HABITAT (Home Assistance Based on the Internet of Things for the AuTonomy of everybody) Project in 2016. He is currently a Junior Assistant Professor with the Department of Electrical, Electronic and Information Engineering "G. Marconi" (DEI), University of Bologna. His research interests include microwave radar systems for biomedical applications, indoor positioning exploiting RFID technologies, near-field and far-field wireless power transfer (WPT), and simultaneous wireless information and power transfer (SWIPT) systems.

Gianluca Cena (Senior Member, IEEE) received the M.S. degree in electronic engineering and the Ph.D. degree in information and system engineering from the Politecnico di Torino, Italy, in 1991 and 1996, respectively. Since 2005, he has been a Director of Research with the National Research Council of Italy (CNR-IEIIT), Italy. He has coauthored about 170 technical papers and one international patent. in his research interests which include wired and wireless industrial communication systems, real-time protocols, and automotive networks. Dr. Cena was the recipient of the Best Paper Award of the IEEE Transactions on Industrial Informatics in 2017 and of the IEEE Workshop on Factory Communication Systems in 2004, 2010, 2017, 2019, and 2020, respectively. He was also a Program Co-Chairperson of the IEEE Workshop on Factory Communication Systems in 2006 and 2008. Since 2009, he has been an Associate Editor for IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.