Computer Networks 253 (2024) 110749

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Perspectives on IoT-oriented network simulation systems

Alberto Gallegos Ramonet [a,*], Tommaso Pecorella [b], Benedetta Picano [b], Kazuhiko Kinoshita [a]

[a] Graduate School of Tech. Ind. and Social Sci. , Tokushima University, 2-1 Minami Josanjima-cho, Tokushima, 770-0814, Japan
[b] Dpt. of Information Eng., Università di Firenze, via di S.Marta 3, Firenze, 50139, Italy

ARTICLE INFO

MSC:
0000
1111

Keywords:
Network simulation
IEEE 802.15.4
IoT
Matter
Communication protocols
Simulation tools
6LoWPAN
Thread
Ns-3

ABSTRACT

The Internet of Things (IoT) paradigm is assumed to be a major component in the present and future Internet, with forecasts claiming a humongous number of devices connected in the near future, and applications fields spanning from agriculture, to healthcare. Despite this, the standardization efforts have not yet resulted in widely adopted standards, and the market is fragmented into multiple solutions both at physical and communication protocol levels. Moreover, IoT systems exacerbate the usual test bed limitations, *e.g.*, scalability (very large number of devices), hardware compatibility, space, and price. Due to the above problems, simulation tools become an extremely interesting tool for studying IoT systems both for academia (new algorithms), standardization (new protocols), and industry (what-if analysis). In this paper we will discuss what are the most relevant features and models that a simulation tool like ns-3 should prioritize to enable the above-mentioned needs from academia, standardization, and industry, and if they are achievable in the short, medium, or long term.

1. Introduction

Fulled by the support of the industry tech giants, the Internet of Things (IoT) promise of all connected "smart" devices society is fast approaching. Nevertheless, at this time, there are still many challenges to overcome before this future becomes a reality.

Simulation and hardware implementations are common tools to develop both applications and communication protocols driving IoT development. In particular, simulation tools play an important role in IoT development, as they can be used for major design phases, namely: (1) protocol development, including academic research and standardization, (2) industrial evaluation of a protocol (what-if analysis), and (3) study of IoT systems in academia.

In the first case, the goal is to test new protocols, new algorithms, etc. that could not be easily analyzed using real hardware, either because the hardware is too costly, or because the simulator allows to analyze scenarios difficult to implement.

In the second case the analysis is more aimed at analyzing the performance of a full system, *e.g.*, would an application drain the batteries of a device if this protocol stack is being used? Also, in this case, the simulation tool can provide access to scenarios difficult to replicate in a testbed.

The third case equally important, allows academic bodies and enables future generations of students to study IoT systems without the costs of a full-scale testbed, which are often beyond acceptable.

Despite the undeniable advantages that simulation tools provide, these tools are not short of challenges, ranging from limited support, interoperability issues, lack of documentation, and narrow capabilities to mention a few.

In this paper, we will discuss the actual limitations of open-source simulation tools concerning IoT standards, and give an overview of what is, in our opinion, the most interesting development and research opportunities. While some of the perspectives discussed in this work are valid for other network simulation tools, we specifically explore the case of the ns-3 network simulator throughout our examples in this work and, provide our perspectives from the point of view of its IoT module maintainers to which the authors belong.

This paper is structured as follows: Section 2 gives an overview of IoT discrete event simulators and the ns-3 network simulator. Section 3 discusses the differences between different simulation and hardware implementations. Section 4 explores some of the challenges faced in IoT network simulations. Our last Section discusses priorities and needs that provide characteristics of future relevant IoT simulation systems. This is followed by our conclusions.

2. Discrete event network simulators for IoT development

Ideally, network simulations should accurately replicate a real network's behavior with an established precision level. Naturally, the

* Corresponding author.
E-mail address: alramonet@is.tokushima-u.ac.jp (A. Gallegos Ramonet).

https://doi.org/10.1016/j.comnet.2024.110749
Received 28 January 2024; Received in revised form 9 July 2024; Accepted 23 August 2024
Available online 24 August 2024

level of precision required can drastically change from user to user. While in some cases users might require simple representations of data transmission from one point to another; other users require a level of granularity that requires the setup of complex simulations of the entire communication process in and out of hardware.

Furthermore, an important consideration in network simulation development is the time cost vs. complexity relationship. A great deal of users are willing to sacrifice simulation precision as long as the creation process is less complex than alternatives. GUI (Graphical User interface) plays a big role in this respect, and many users will favor a network simulator that uses GUI over a text-based interface. Unfortunately, many open network simulation tools are text-based only with limited graphical options. A large portion of users find text-based simulators to have steep learning curves as they require both programming experience and computer network knowledge to use.

Licensed paid network simulations, on the other hand, include GUI options but are often not open source, and even in the cases where the source is included, the simulator modules cannot be extended, studied, or modified in any shape or form without the licenses mentioned above. This is a severe limitation for most researchers.

When choosing a network simulator, users must weigh these limitations and choose the best option for each use case.

Without loss of generality, in the following, we will refer mainly to ns-3, as the authors have an extensive knowledge of the simulator, and are active maintainers of some of its models. However, the points that will be discussed can be applied to different systems.

2.1. The ns-3 network simulator

Ns-3 is a discrete event network simulator that supports the simulation of different types of computer networks. Ns-3 implements IoT-related networks mainly in its Lr-WPAN and 6LoWPAN modules. Lr-WPAN has support for a single simulated 2.4 GHz Band with an O-QPSK modulation. Ns-3's Lr-WPAN module also supports a MAC layer (IEEE 802.15.4-2003-2011) with network initialization options (scanning and association) and 2 supported transmission modes: beacon and beaconless mode. Both ns-3's Lr-WPAN PHY and MAC implement most of the primitives described by the standard (2011) which are used to directly interface with these layers.

Ns-3's Lr-WPAN module is currently missing support for Guaranteed Time Slots (GTS) and has only partial support for indirect transmissions (present in the association procedure). Moreover, because ns-3 only implemented the standard until the 2011 revision, advanced MAC behaviors such as TSCH (Time slotted channel hopping) and DSME (Deterministic and synchronous multi-channel extension) are not included. Likewise, capabilities to provide the RSSI (Received signal strength indication) are not supported. All of these features were introduced with the 2015 revision of the standard. On the physical layer side, ns-3 only supports a 2.4 GHz ISM band with an O-QPSK modulation error model. While the standard covers a wide variety of modulations and bands [1], these are not typically covered by most hardware implementations in the market but are a point of improvement that ns-3 can look forward to in the future.

Ns-3's 6LoWPAN model can be used in conjunction with either Lr-WPAN or other standards like WiFi. 6LoWPAN is necessary to use IPv6 over Lr-WPAN, due to IPv6's maximum transmit unit (MTU) requirements. Moreover, it alleviates some shortcomings of Low-Power and Lossy Networks (LLN), like short packets leading to excessive overhead (by implementing compression) and simple support for multicast and mesh routing. Even though 6LoWPAN was developed for LLNs, it can be used also on other network kinds.

Besides Lr-WPAN or 6LoWPAN, ns-3 does not have official support for other IoT-oriented protocol stacks (network or application layers). However, 3rd parties have made some progress in this respect as well as modules for other IoT standards such as LoRA.

While ns-3 source code is highly documented and organized, its lack of GUI tools to generate network scenarios could be considered by many a major drawback. This is especially true for beginners, who might find it a daunting task to code network scenarios in C++ while also learning the basics of computer networking. The importance of GUI development and present capabilities of ns-3 is discussed in detail in Section 5.1.

In its present state, ns-3's LR-WPAN and 6LoWPAN modules are not able to communicate with external hardware. However, future integration is possible. Current ns-3 hardware emulation support and the specific case of IoT hardware emulation potential are discussed in Section 5.2.

3. IoT protocols: standards vs. market adoption

The number of protocols that are and can be used by IoT systems is large and still growing. Often we refer to transmission protocols as either wireless personal area networks (WPAN) or low power wide area networks (LPWAN), depending on the radio range of the technology. Examples of such protocols are the widely used IEEE 802.15.4 for WPAN and LoRA/LoRAWAN for LPWAN. In the following sections, without loss of generality, we will refer to these two protocols. Naturally, they are not the only protocols available, notable mentions include Bluetooth (IEEE 802.15.1) and WBAN (IEEE 802.15.6) for WPAN networks and SigFox for LPWAN networks. However, due to space constraints, they will not be discussed in this work.

3.1. LoRA and LoRAWAN

Long Range (LoRa) is a physical layer (PHY) developed by Semtech company and based on the Chirp Spread Spectrum (CSS) modulation. It operates on the unlicensed frequency bands 868 MHz and 915 MHz. Its promise of long-range communication with low energy consumption makes it an attractive offer for many applications in open-area environments. However, due to the same long-range coverage and frequency characteristics, its performance can drastically vary in situations where many nodes exist or in scenarios with many obstacles such as buildings that can cause signal degradation due to attenuation and fading effects. Its link layer (*i.e.*, MAC layer) was developed by the LoRA Alliance and is known as LoRAWAN. LoRAWAN uses the medium access method known as ALOHA. LoRA/LoRAWAN does not have any official stack that supports mesh networks. However, numerous works exist in the literature addressing this point [2]. Being relatively new when compared to other IoT protocol stacks, fewer simulation options exist. Models for both OMNet++ [3] and ns-3 [4,5] simulators exist, but they do not exist in an official capacity and are only externally maintained.

3.2. IEEE 802.15.4 std.

The IEEE 802.15.4 std. is the low-rate wireless personal area network (Lr-WPAN) standard and is currently the de facto standard used in IoT (Internet of Things) home applications. The standard describes a physical layer (PHY layer) and a link layer (a.k.a MAC layer). While the standard defines these 2 layers, users are free to use any supported protocol stacks on top of these layers. To date, this standard has issued 5 major revisions (2003, 2006, 2011, 2015, 2020). These revisions describe the support for a wide range of band frequencies and modulations. However, IEEE 802.15.4 capable devices found in the market are mostly constrained to the 2006 revision of the standard and the 2.4 GHz ISM band (250 kbps O-QPSK). Our survey on manufacturers of this standard indicates that adoption of newer revisions is slow and typically not adopted until many years after their release, therefore, it is not uncommon to find decade-old or older standards implemented even in devices recently introduced to the market.

Table 1
IEEE 802.15.4 std. compliant hardware and their supported standard.

Manufacturer	Model	IEEE 802.15.4 Standard Rev.
NXP [6]	JN516x/JN517x series	2006
Texas Instruments [7]	CC253x/CC263x series	2006
Nordic Semiconductor [8]	nRF528xx series	2006
NXP [9]	K32W061/41	2011
NXP [9]	JN518x series	2011
Expressif [10]	ESP32-H2	2015
Silicon Labs [11]	EFR32MGxx series	2012 (g)

Table 2
Network Simulators and their supported IEEE 802.15.4 implementations.

Network simulator	Version	IEEE 802.15.4 Standard Rev.	Actively supported	Open source	Doc
ns-3 [12]	3.42	2011	Yes	Yes	Web PDF
ns-2 [13]	2.35	2003	No	Yes	Web
Qualnet [14]	9.3.0	2006	Yes	Yes[a]	PDF
Omnet++ [15]	4.6	2006	No[b]	Yes	PDF doc
Opnet [16]	18.9.0	Not disclosed	Yes	Yes[a]	Web

[a] Requires license fees.
[b] Omnet++ is actively supported but the module which implements IEEE 802.15.4 (Castalia) is not.

Table 1 shows a compilation of released IEEE 802.15.4 capable hardware devices. Similarly, Table 2 shows a comparison of simulators similar to ns-3 and their supported IEEE 802.15.4 revisions. From the information shown in these tables, the most adopted revision is the 2006 revision closely followed by the 2011 revision. As shown in Table 1, even with devices considered state of the art like the ESP32-H2 released in 2023, the latest adopted revision was the 2015 revision. The definition of revision used is often overlooked in evaluations, but the difference between revisions is an important characteristic because it denotes the capabilities of the device or the simulation. For instance, there are significant differences [17] among 2006, 2015 revisions and the 2012(g) amendment.

From the standard point of view, revisions are incremental. For example, unless specifically stated, a 2011 revision will typically include features described in both 2003 and 2006 revisions (with minimal changes and a few deprecated content). On the other hand, simulations and hardware implementations do not always include all the features described by the standard. Our survey indicates that the level of completeness drastically varies in both hardware and simulation implementations. Overall, most implementations support the 2.4 GHz ISM band and a few of them also support 868 MHz and 915 MHz bands. Other regional, industrial, or medical bands are typically not included in devices meant for smart home devices. Likewise, MAC modes such as the beacon-enabled mode are often not included in implementations. This is mainly because higher layer protocol stacks like the Thread and Zigbee Pro stacks do not make use of this MAC mode or the support of additional PHY bands[1] which are out of the scope of these standards.

3.3. The Thread standard

Thread is an open standard for low-power low-data rate devices and it is designed to provide security and IPV6 network connectivity options to IEEE 802.15.4 devices. More specifically, Thread is a combination of multiple communication protocols that complement IEEE 802.15.4 devices' capabilities, hence is often referred to as the Thread stack. When IEEE 802.15.4 was conceived it was meant to be used in mesh networks (multi-hop networks), however, IEEE 802.15.4 devices themselves cannot form mesh networks, they require the use of higher layer protocol stacks to do so. In IEEE 802.15.4, all devices are part of a Personal Area Network (PAN) and must be organized in a coordinator-end device relationship. Coordinators provide essential services to end devices such as assigning short MAC addresses and access to join a particular PAN. While the IEEE 802.15.4 MAC layer typically provides these services, in Thread, these MAC services are not used but are provided by Thread instead. The Thread protocol stack uses protocols that have not been standardized by IETF or with modifications that make them non-standard. A significant example of this is the modified version [19] of the Mesh Link Establishment (MLE) protocol [20] which is used to provide address assignment and additional services such as neighbor tracking and administration of asymmetric links. Furthermore, Thread uses 6LoWPAN header compression but with DHCPv6 for IPV6 address assignment instead of 6LoWPAN-ND [21–23]. The IPV6 routing protocol is custom-built and is loosely based on the routing information protocol (RIP) [24,25].

Google's OpenThread [26] is arguably the most popular implementation of Thread. Originally it was designed to work with IEEE 802.15.4-2006 compliant devices, but devices such as JN5189, nRF528xx, and ESP32-H2 which run newer revisions of the standard work with the OpenThread as described by its documentation. OpenThread can be used with a System-on-Chip (SoC) design, meaning that the complete stack and applications are embedded into an integrated circuit of an IEEE 802.15.4 device. Alternatively, OpenThread has support for co-processor designs: (1) The radio co-processor (RCP) design in which the application and the Thread stack are running on a separate processor and communicate with an IEEE 802.15.4 device via UART or SPI serial protocols, (2) The network co-processor (NCP), where the OpenThread stack is also embedded into a chip and communicates with an application in the host processor via serial communication. OpenThread designs are summarized in Fig. 1.

To simulate Thread on ns-3 there are two future approaches. The first one is to integrate OpenThread and ns-3, using ns-3 to simulate IEEE 802.15.4. The second one is to add the required protocol models to ns-3 (*i.e.*, CoAP, MLE, RIP).

The first alternative seems to be the most straightforward, but it is also the most limiting. The limitations are relative to the capability to make slight changes in the protocol, like substituting DHCPv6 with 6LoWPAN-ND, or extending the base Thread routing protocol.

Adding all the required protocols to ns-3 is not straightforward but is conceivable, with the main complexity being represented by accessing the protocols' formal definitions.

3.4. Zigbee

The CSA's (Connectivity Standards Alliance) *Zigbee* protocol stack was specifically designed to be used along IEEE 802.15.4. It complements many of its shortcomings, notoriously, neighbor discovery capabilities, security, and mesh routing. Unlike similar protocol stacks like Thread, Zigbee is not IP-dependent, instead, it uses a combination of 16-bit and 64-bit addresses (short and extended addresses) already present in the definition of IEEE 802.15.4.

The specification of Zigbee has seen many changes throughout its more than 20 years of history. It is precisely this longevity its major asset, making Zigbee one of the most proven and extended IoT-oriented full stacks in the market. Zigbee can be roughly divided into 3 parts: The network layer (NWK), the application support layer (APS), and the application layer (APL).

- The NWK complements the MAC layer association capabilities (network bootstrap and joining) and adds tree and mesh routing capabilities. The mesh routing is loosely based on the Ad hoc On-Demand Distance Vector Routing (AODV) but with enhanced support for neighbor discovery and link quality management capabilities among others.

[1] Zigbee 3.0 R23 revision (2023) [18] added for the first time support for the long released 868 MHz and 915 MHz bands but from our review there were no Zigbee Pro R23 devices in the market that make use of these bands.

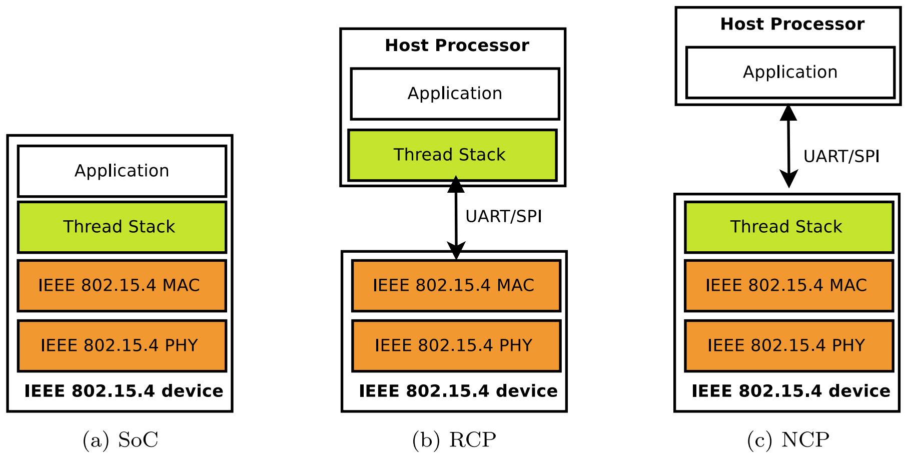

Fig. 1. OpenThread Designs.

- The APS is responsible for the management of application bindings, messages between devices, mapping between extended and short addresses, fragmentation, and reassembly.
- The APL defines the environment in which applications are hosted on devices. It relies on profiles, clusters, and attribute modes. It is this layer that later on will form the basis of what is known as the *Matter* application layer. Both Zigbee APL and Matter share a common creator in the CSA and common concepts like the use of application profiles. However, both of these layers work on top of different protocol stacks and are not compatible with each other.

ZBOSS [27] is the only licensed complete open-source implementation of the Zigbee stack. Its version 1.0 is freely accessible to the public while subsequent releases are only available to members of the ZBOSS public initiative which requires a paid membership. The ZBOSS commercial version of the stack is popular among IoT device vendors like Nordic Semiconductors and Espressif. Vendor-specific binaries of the stack can be found on the websites of these ZBOSS public initiative members. Other Open-source alternatives include Zigpy [28], but it might not be suitable for every deployment because is an RCP design that requires Python.

The support of Zigbee on discrete event simulations is very limited. This is in part due to its restrictive license. It is important to notice that the terms Zigbee, AODV, and IEEE 802.15.4 are often used in the literature indistinctly. However, as described in this document, Zigbee is not IEEE 802.15.4 nor AODV. In the present document, we consider a Zigbee-capable simulator that implements at least a Zigbee NWK layer. In the literature, some works have reported using Zigbee simulation results using Riverbed's Opnet simulator [29,30]. Likewise, the 2006 Specification of Zigbee was implemented in the now-discontinued ns-2 simulator. Currently, there is no official implementation of Zigbee in ns-3 simulator.

3.5. The Matter standard

Matter [31] is a general-purpose application layer standard by the CSA, the same alliance that introduced the Zigbee Protocol. The objective of Matter is to create a common API (application programming interface) that helps to create applications for a diverse range of communication hardware technologies (Lr-WPAN, WiFi, Bluetooth LE). Before Matter, each vendor had to introduce their proprietary application layer API. This created incompatibility between "smart" devices from different vendors and created a great deal of frustration among IoT application developers who had to maintain their applications for each hardware variation. In Matter, all devices talk the same language therefore, compatibility is increased and development time is reduced without major issues.

On the downside, Matter fails to address a significant backward compatibility issue. What to do with the millions of existing IoT devices already deployed? Matter is an application layer that depends on various underlying protocol stacks to function (Fig. 2). This combination of protocols might not be interoperable with many of the existing IoT devices in the market.

In IEEE 802.15.4 devices, Matter must be combined with the Thread protocol stack. That means that any device using existing alternative stacks like Z-wave or Zigbee stacks will not work with Matter. Given the popularity of Zigbee as one of the IoT pioneering "complete stack solutions" and with millions of devices in the market this is not a small problem to have. The CSA has proposed the use of bridge devices that can talk both Thread and Zigbee. However, the CSA only provides soft guidelines of how these bridge devices should work and require additional hardware. This opens yet another front line of potential interoperability issues and puts the support of legacy Zigbee devices at risk even with the optimistic Zigbee 3.0 revision R23 [18] released at the start of 2023. Manufacturers can provide firmware updates to port existing devices from a Zigbee/Z-wave stack to a Thread stack, however, with no financial motivations or constrained by the processing or memory capabilities of the deployed devices, in most cases these solutions are unlikely to happen or at best relegated to a network co-processor (NCP) solution.

For home use cases, Matter increases the portability of applications for smart devices but it will not serve all types of applications. Matter applications require a considerable overhead to work (Matter + Thread + TCP/IP), this might not be the best approach for devices that have only a few kb of memory to spare and limited battery life. After all, the IEEE 802.15.4 standard was intended for low data rates and memory-constrained devices from which in many cases the complete application and protocol stack must be embedded into the chip. Privacy and security are also a concern. Security is extensively covered in Thread but comes with the aforementioned overhead and in many cases users might desire to remain independent from an IP network, which is arguably the safest and most private approach.

To summarize, Matter is a solution for IoT home devices and consumers can expect to see more of it in the future due to the big tech companies support. Unfortunately, Matter has a substantial overhead that might not suit every solution in the market. Because of this and other limitations described, it will not serve most legacy devices, or provide a common application layer solution for non-IP-dependent networks.

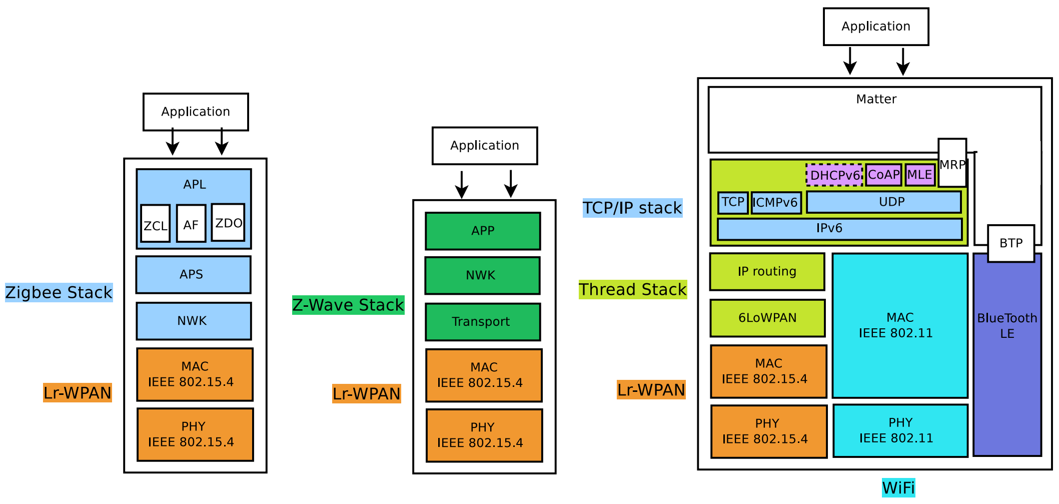

Fig. 2. IoT protocol stacks comparison.

4. Network simulations challenges

Preview sections show that the reality of the standardization for IoT systems does look like a famous comic strip.[2] Actual or proposed network stacks range from non-IP stacks like Zigbee, Z-Wave, or LoRaWAN to IPv6-oriented, like Thread and Matter.

The disparity found between different user requirements places a strain on simulation developers, who must find a balance that fulfills the needs of academia, industry, and standardization bodies. The following subsections briefly describe important challenges faced in IoT network simulation system development.

4.1. Physical and MAC layers (L1 and L2)

Physical layer simulations can be done considering the low level of granularity taking place in this layer (*e.g.*, encoding and decoding of frames and considering signal behaviors). These types of simulators are known as link-layer simulators and are widely used by researchers and standardization bodies. Link-layer simulators are ideal for representing data transmissions between a couple of devices. However, because computational requirements are incompatible with network-level simulations, discrete event simulators are better suited for these tasks. In such simulators, the physical layer is often simulated through statistical properties rather than simulating the exact signals transmitted or received. Toward this end, most discrete event simulation tools can simulate different scenarios using channel propagation models, interference models, etc. These are easily extendable and usually do not represent a major issue. It is worth mentioning, however, that these propagation models are typically developed as discrete approximations and do not mimic all wireless conditions present on real devices. Representation of a full set of these conditions is still a major challenge [32].

The medium access control (MAC) layer is simulated as accurately as possible. However, it represents a peculiar element because the amount of possible ‘customization’ in a real device is usually minimal. As a matter of fact, the MAC level is usually implemented in the device, and developers have no access to the source code, which makes it difficult to propose changes or improvements to the MAC layer in any significant way. Likewise, this directly impacts the level of precision with which a hardware-specific MAC layer can be accurately reproduced in a simulation.

Academia and standard bodies can be interested in analyzing changes in the MAC level (*e.g.*, new techniques for collision avoidance), and simulations are essential tools in these cases. Nevertheless, the aim of a simulator should be to provide the user with an implementation of the ‘base’ standard, possibly of the latest version.

4.2. Network and transport layers (L3 to L6)

As shown in Section 3, Fig. 2, protocol stacks can be quite different from each other. As a consequence, there are two different sets of needs.

4.2.1. Single-choice stacks

We call ‘single-choice’ stacks those where the protocol alternatives are limited or non-existent. Examples are Z-Wave, LoRaWAN, Zigbee, etc. In these stacks, implementation elements are mandated by specifications, and there is little to no room for changes. The goal is to be as close as possible to the specification APIs and conventions in such a manner that the implementation can be used to evaluate application-level performances and proposals for modifications to the specification.

We shall note that stacks like Thread, which can use alternative IPv6 elements, can be considered ‘single-choice’ because it is not possible to modify elements such as routing or neighbor discovery protocols without deviating from the specification.

Nevertheless, it is useful to provide some degree of flexibility to allow research and standardization alike.

4.2.2. Multi-choice stacks

Multi-choice stacks are the ones based on IPv6 standards (or equivalent) where the user is free to customize several parts of the stack, *e.g.*, the routing type.

Multi-choice stacks are primarily used for research purposes, or to build custom networks. The goal of the simulators in this case is also to help identify the pros and cons of choosing alternatives, like routing or transport protocols. Toward this end, a simulator should allow a user to have a set of alternative protocols, or to allow the user to (easily) integrate their models in the system.

4.3. Application layer (L7)

The application-level protocols are, usually, a problematic element for simulations, mainly because they also have to consider human interactions. E.g., a web client should mimic the user behavior, which is (usually) complex.

[2] https://xkcd.com/927/

The solution is to use 'standardized' models for application-level behavior, often derived from standardization bodies. These can be described as user-level interactions (*e.g.*, HTTP requests and the replies' size) or traffic flows (*i.e.*, UDP or TCP traffic generation).

5. Development priorities and opportunities

In previous sections, we presented an outline of some of the most popular commercially available IoT standard protocol stacks. Additionally, we draw comparisons between hardware implementations and some of their simulated counterparts. Differences in these implementations highlighted some of the many challenges that IoT-simulated systems must overcome. In this section, we discuss what, in our opinion, are development priorities, and research opportunities to improve future IoT-oriented network simulation systems. The ns-3 simulator is used as a base to present our recommendations (Fig. 4). While the level of importance may vary for similar projects, we believe that these guidelines are general enough to identify the priorities in any simulation system.

5.1. Accessibility and user interfaces

The addition of novel communication protocols offered by IoT network simulation systems is an important part of the existing offerings. However, there is a pressing element that deserves more attention from the community in the existing simulation systems: user interaction.

Accessibility elements in network simulators are, in our opinion, currently the main need to sustain a great IoT network simulation tool. Regardless of whether the simulation tool is used to perform a "what-if" scenario, enhance protocol performance, or for didactic purposes, the simulation should be easy to set and visualize. In this sense, the development of a more varied and rich GUI capable of setting complex simulated scenarios could greatly benefit current network simulation systems, which are traditionally heavily text-driven.

Network simulators are often considered to have steep learning curves because they require knowledge of network standards and a variety of programming and scripting languages to use. GUI can minimize this burden for newcomers. Furthermore, GUI provides a means to minimize the setting of menial but time-consuming tasks such as topology shapes and protocol stack configurations. GUI should be seen as a complement to text-based simulations, not a replacement. This is analogous to tools such as Unreal Engine, widely used in the entertainment sector. This tool combines both the benefits of GUI interfaces (*e.g.*, unreal engine blueprints) and interactions with diverse programming languages to create complex simulated environments.

The lack of GUI assistant tools in network simulations should not imply that text-driven simulations do not offer facilities to quickly set network simulations. For instance, the ns-3 network simulator introduced the use of *helpers* since its beginnings. Helpers are small pieces of code, and their role is to simplify the module configuration, taking care of the class setup and initialization of different properties. Traditionally, each ns-3 module provides a set of one or more of these helpers.

In the particular case of IoT, a network simulation system could benefit from the simulation of complex indoor scenarios considering walls. Toward this end, it would be useful to enhance the capabilities of existing tools and enable their models to import floor plans with node placements, which could be later exported in SVG files. Another useful feature is the capability to infer and draw network topologies. This is a complex problem, as topology can change over time and is dependent on the protocol stack: physical level, IP level, routing level, application level, etc.

Moreover, a GUI should be capable of reading/writing the simulation parameters and configuration to an easily readable format (*e.g.*, ASCII, JSON, or XML files) to allow variations of a particular scenario. In the specific case of ns-3, this is already implemented in the feature known as *ConfigStore* but this could be further improved and used as part of a unified GUI.

Creating GUIs for existing text-based network simulators is no easy task. To create simulated scenarios using a GUI, a descriptive language is necessary to serve as a bridge between the graphical interfaces and the text-based simulation. One example of such languages includes the network description (NED) language used by OMNet++. The amount of data and simulation details that should be written to a network description file is highly subject to debate and can range from a representation of simple protocol parameters and node positions to a full representation of networks, including the device's relationships, protocols per node, stack configuration parameters, etc. In a simulation system, it is advisable to aim for a balance between completeness and usability. Usability has the primary goal of allowing a user to effortlessly run common tasks, like slight variations on the scenario, to perform Monte Carlo analysis.

In this regard, GUI proposals [33–36] have been made to ns-3 in the past, but none of them have been officially adopted because of problems related to long-term maintenance, scope, and complexity. Nevertheless, officially maintained visualizers [37–39] do exist and provide a soft level of visualization capabilities to ns-3. It is worth noting that, unlike GUI simulation creation tools, visualizers are unable to graphically create simulations from scratch, only to show them after the simulation has taken place. To succeed where many other proposals have not reached an audience, future GUI tool proposals must take into consideration ns-3 present features, keep a balance between depth and complexity, and finally, make the appropriate considerations for its maintenance.

5.2. Emulation and reproducibility

As a second level of importance, we place the capability of IoT network simulation systems to interact with hardware implementations. Network simulations should represent the behavior of existing hardware implementations, however, this level of realism is hard to achieve without close comparisons to real devices. A practical way to achieve this reproducibility requirement is by crossing over the capabilities of the simulator with those of hardware implementations, in essence, combining simulator and hardware implementation stacks. In simulation systems, these capabilities are commonly referred to as emulation capabilities. Many proposals have been made to advance the emulation capabilities of network simulators such as ns-3 [40,41] and OMNet++ [42].

For IoT devices, making these crossovers presents unique challenges.

First, IoT models in discrete event simulators are designed to mimic standards or specifications descriptions. In many cases, the configuration values used in these descriptions might not be ideal in some real deployment environments or altogether not used by hardware implementations. Furthermore, hardware implementations are tailored to specific devices and it is not uncommon to find deviations or additions to the standard. Naturally, these small changes create inconsistencies across devices which affects interoperability and crossover usability.

Second, while standard specifications are often open to the public, the protocol implementations found in hardware are mostly closed-source. These IoT device-specific stacks are with a few exceptions, non-interchangeable across devices from different vendors, and often use monolithic designs. This makes it difficult for simulators to reproduce results from specific hardware as looking underneath these implementations is often not possible.

Nevertheless, simulation developers must overcome these challenges and offer IoT simulation tools that resemble the implementation used in hardware.

In ns-3, emulation capabilities are typically provided by a generic device known as *FdNetDevice* capable of reading and writing from a Linux file descriptor. This file descriptor can be associated with an

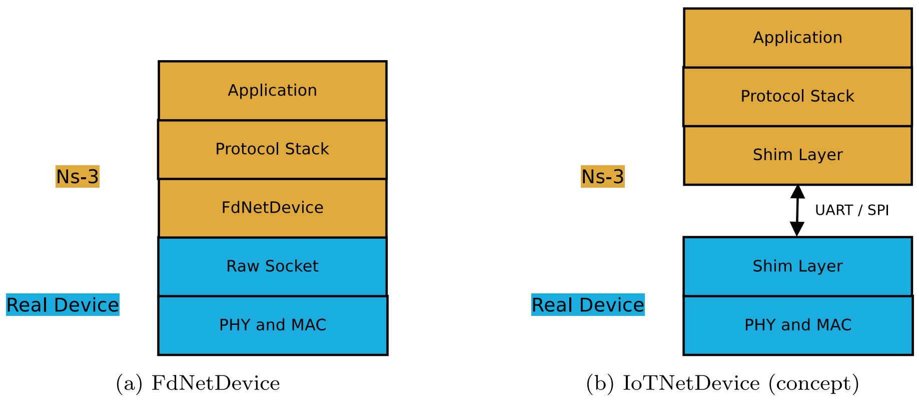

Fig. 3. Ns-3 emulation capabilities examples.

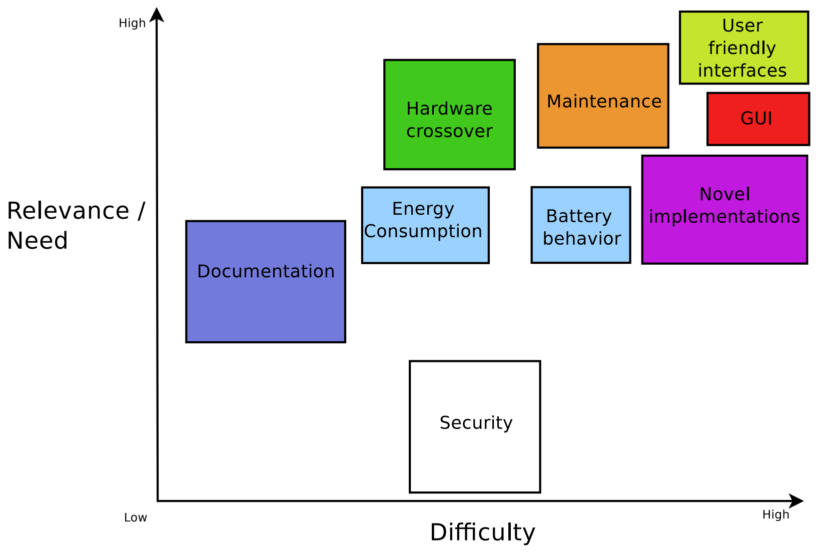

Fig. 4. Ns-3 IoT simulation system priorities and needs.

underlying packet socket to provide connectivity with the host system's real device. However, with a few exceptions (*e.g.*, Linux-WPAN [43]) this file descriptor emulation method is not possible for IoT devices that have a complete or a piece of a stack in an MCU instead of the Linux kernel. For those cases, it is possible to provide emulation capabilities and allow ns-3 to send data to real devices via UART or SPI protocols. This is in essence, the same approach used by the Thread protocol explained in Section 3.3. While this feature is not present in the current release of ns-3 (v3.42) its future implementation is possible by providing a shim layer to bridge a host computer and the MCU devices (Fig. 3).

5.3. Maintenance

As a third level of importance, we consider the simulation codebase evolution. Many implementations of IoT protocol simulations exist. However, not all implementations are maintained or grow after deployment. New specifications and extensions to existing communication protocols are introduced every day but reliable IoT simulations are many years behind and more often than not, stagnant. This puts many IoT simulations found in the literature under the category of abandonware, where implementation is done once and maintenance of the project is short-lived or never considered.

When choosing a simulation system, users should make careful considerations and be aware of the shortcomings in documentation, activeness in development, and the version of the protocol simulated. As described in Section 3, it is often the case that available IoT simulations and hardware from both open-source and commercial simulators are either incomplete or outdated when compared to the most recent specifications.

A great IoT simulation system like any great piece of software should in our opinion, contain models that are actively developed, maintained, and consistently documented. This is not a simple problem to solve, as most of the IoT simulations are done by volunteer work and many authors find little to no motivation to maintain their code.

5.4. Energy modules development

Energy evaluation plays a crucial role in IoT network simulations. Unlike many other networks, IoT networks are predominantly batteryoperated, imposing constraints on data transmission and operational energy usage. A meticulous examination of energy consumption becomes imperative for the development of energy-efficient communication protocols or the identification of patterns that can minimize a device's energy footprint. Achieving this necessitates intricate models that accurately capture both the consumption patterns of transceivers and the non-linear discharge behavior of batteries [44–46], with the latter aspect often being overlooked.

5.5. Security

Finally, a significant difference between simulated and hardware IoT implementations is that security is often treated as a top priority in hardware implementations, taking as much as half of the descriptions of a specification. However, from the point of view of an IoT simulation, this implementation is often the least important element.

Performing a real security process (*e.g.*, cryptography algorithms) in a simulation adds unnecessary complexity to the simulated system and increases its processing power requirements. Therefore security components are often omitted in IoT simulations. However, we argue that a great IoT simulation system should support the representation of the overhead [47] caused by the security process:

- Delay caused by the processing of coding and decoding packets.
- The packet overhead resulting from the variations in packet size when security options are enabled.

6. Conclusions

In this paper, we analyzed the IoT networking challenges and how network simulations can be useful to help academy, standardization, and industry to build a solid next-gen set of protocols specifically tailored for IoT devices.

The standardization efforts are still in their early stages, and market tensions are still at play (sometimes undermining the standardization efforts). Hence, a solid and flexible simulation system is needed to help the interested parties in performing the right choices, or quickly spot the possible elements to be optimized.

We also highlighted what are, in our opinion, the most promising protocols to model in ns-3, and what are the gaps to fill in order to have a tool that is powerful, easy to use, and flexible.

CRediT authorship contribution statement

Alberto Gallegos Ramonet: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Tommaso Pecorella:** Writing – review & editing, Writing – original draft, Supervision, Software, Project administration, Investigation, Conceptualization. **Benedetta Picano:** Validation, Formal analysis, Data curation, Conceptualization. **Kazuhiko Kinoshita:** Validation, Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] A.G. Ramonet, T. Noguchi, IEEE 802.15. 4 now and then: Evolution of the LR-WPAN standard, in: 2020 22nd International Conference on Advanced Communication Technology, ICACT, IEEE, 2020, pp. 1198–1210.

[2] J.R. Cotrim, J.H. Kleinschmidt, LoRaWAN mesh networks: A review and classification of multihop communication, Sensors 20 (15) (2020) http://dx.doi.org/10.3390/s20154273, URL https://www.mdpi.com/1424-8220/20/15/4273.

[3] M. Slabicki, G. Premsankar, M. Di Francesco, Adaptive configuration of lora networks for dense IoT deployments, in: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–9, http://dx.doi.org/10.1109/NOMS.2018.8406255.

[4] D. Magrin, M. Centenaro, L. Vangelista, Performance evaluation of LoRa networks in a smart city scenario, in: 2017 IEEE International Conference on Communications, ICC, 2017, pp. 1–7, http://dx.doi.org/10.1109/ICC.2017.7996384.

[5] L. Vangelista, A. Cattapan, Extending the lora modulation to add parallel channels and improve the LoRaWAN network performance, in: 2021 11th IFIP International Conference on New Technologies, Mobility and Security, NTMS, 2021, pp. 1–5, http://dx.doi.org/10.1109/NTMS49979.2021.9432659.

[6] NXP, IEEE 802.15.4 Stack user guide, 2024, https://www.nxp.com/products/wireless-connectivity/zigbee/ieee-802-15-4-for-jn516x-7x:IEEE802.15.4, (Accessed January. 17. 2024).

[7] Texas Instruments, TI MAC software stack, 2023, https://www.ti.com/tool/TIMAC, (Accessed December. 15. 2023).

[8] Nordic Semiconductor, Product Specifications, 2024, https://infocenter.nordicsemi.com/index.jsp, (Accessed January. 17. 2024).

[9] NXP, JN-518x DataSheet, 2024, https://www.nxp.com/products/wireless-connectivity/thread/jn5189-88-t-high-performance-and-ultra-low-power-mcus-for-zigbee-and-thread-with-built-in-nfc-option:JN5189_88_T, (Accessed January. 17. 2024).

[10] Expressif, ESP32-H2 Datasheet, 2023, https://www.espressif.com/sites/default/files/documentation/esp32-h2_datasheet_en.pdf, (Accessed December. 15. 2023).

[11] Silicon Labs, EFR32MG13 Mighty Gecko Multi-Protocol Wireless SoC Family Data Sheet, 2023, https://www.silabs.com/documents/public/data-sheets/efr32mg13-datasheet.pdf, (Accessed December. 15. 2023).

[12] Nsnam, Ns-3 network simulator, 2023, https://www.nsnam.org/, (Accessed December. 15. 2023).

[13] Nsnam, Ns-2 network simulator, 2023, https://www.isi.edu/nsnam/ns/index.html, (Accessed December. 15. 2023).

[14] Keysight, QualNet Network Simulator, 9.3.0 Programmers Guide, Sensor Networks Library, 2023, https://www.keysight.com, (Accessed December. 15. 2023).

[15] T. Boulis, D. Pediaditakis, Castalia 3.2 User Manual, 2024, https://github.com/boulis/Castalia, (Accessed January. 17. 2024).

[16] Riverbed Opnet Modeler, Online documentation, 2023, https://www.riverbed.com, (accessed December. 15. 2023).

[17] A.G. Ramonet, T. Noguchi, IEEE 802.15. 4 now and then: Evolution of the LR-WPAN standard, in: 2020 22nd International Conference on Advanced Communication Technology, ICACT, IEEE, 2020, pp. 1198–1210.

[18] Connectivity Standards Alliance, Zigbee Specification R23, 2023, https://csa-iot.org/all-solutions/zigbee/, (Accessed December. 15. 2023).

[19] Thread Group, Thread Network Fundamentals V3.1, 2023, https://www.threadgroup.org/, (Accessed December. 15. 2023).

[20] IETF, Mesh Link Establishment, 2024, https://datatracker.ietf.org/doc/html/draft-ietf-6lo-mesh-link-establishment-00, (Accessed January. 04. 2024).

[21] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, Transmission of IPv6 packets over IEEE 802.15.4 networks, RFC 4944 (Proposed Standard), RFC Editor, Fremont, CA, USA, 2007, http://dx.doi.org/10.17487/RFC4944, Updated by RFCs 6282, 6775, 8025, 8066, URL https://www.rfc-editor.org/rfc/rfc4944.txt.

[22] S. Chakrabarti, E. Nordmark, C. Bormann, Neighbor discovery optimization for IPv6 over low-power wireless personal area networks (6LoWPANs), in: Z. Shelby (Ed.), RFC 6775 (Proposed Standard), RFC Editor, Fremont, CA, USA, 2012, http://dx.doi.org/10.17487/RFC6775, Updated by RFC 8505, URL https://www.rfc-editor.org/rfc/rfc6775.txt.

[23] E. Nordmark, S. Chakrabarti, C. Perkins, Registration extensions for IPv6 over low-power Wireless Personal Area network (6LoWPAN) neighbor discovery, in: P. Thubert (Ed.), RFC 8505 (Proposed Standard), RFC Editor, Fremont, CA, USA, 2018, http://dx.doi.org/10.17487/RFC8505, URL https://www.rfc-editor.org/rfc/rfc8505.txt.

[24] C. Hedrick, Routing information protocol, RFC 1058 (Historic), RFC Editor, Fremont, CA, USA, 1988, http://dx.doi.org/10.17487/RFC1058, Updated by RFCs 1388, 1723, URL https://www.rfc-editor.org/rfc/rfc1058.txt.

[25] G. Malkin, RIP version 2 - carrying additional information, RFC 1723 (Internet Standard), 1994, http://dx.doi.org/10.17487/RFC1723, Obsoleted by RFC 2453, URL https://www.rfc-editor.org/rfc/rfc1723.txt.

[26] Google, Openthread, 2024, https://openthread.io/, (Accessed January. 17. 2024).

[27] DSR, Z-boss stack, 2024, https://dsr-zboss.com/, (Accessed January. 17. 2024).

[28] Python based Zigbee Protocol stack, 2024, https://github.com/zigpy/zigpy, (Accessed April. 23. 2024).

[29] S. Vançin, E. Erdem, Design and simulation of wireless sensor network topologies using the ZigBee standard, Int. J. Comput. Netw. Appl. (IJCNA) 2 (3) (2015) 135–143.

[30] G. Spanogiannopoulos, N. Vlajic, D. Stevanovic, A simulation-based performance analysis of various multipath routing techniques in ZigBee sensor networks, in: Ad Hoc Networks: First International Conference, ADHOCNETS 2009, Niagara Falls, Ontario, Canada, September 22-25, 2009. Revised Selected Papers 1, Springer, 2010, pp. 300–315.

[31] CSA, Matter Core Specification 1.0, 2023, https://csa-iot.org/developer-resource/specifications-download-request/, (Accessed December. 15. 2023).

[32] H. Fontes, R. Campos, M. Ricardo, A trace-based ns-3 simulation approach for perpetuating real-world experiments, in: WNS3 '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 118–124, http://dx.doi.org/10.1145/3067665.3067681.

[33] P.D. Barnes, B.B. Abelev, E. Banks, J.M. Brase, D.R. Jefferson, S. Nikolaev, S.G. Smith, R.A. Soltz, Integrating ns-3 model construction, description, preprocessing, execution, and visualization, in: Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, SimuTools '13, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2013, pp. 176–181.

[34] M. Hawa, A graphical user interface for the ns-3 simulator, in: Proceedings of the 12th International Conference on Computer Modeling and Simulation, ICCMS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 159–163, http://dx.doi.org/10.1145/3408066.3408088.

[35] P. Kourzanov, Live network simulation in julia: design and implementation of LiveSim.jl, in: Proceedings of the 2018 Workshop on Ns-3, in: WNS3 '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 30–36, http://dx.doi.org/10.1145/3199902.3199908.

[36] S. Si-Mohammed, M. Janumporn, T. Begin, I.G. Lassous, P. Vicat-Blanc, SIFRAN: evaluating IoT networks with a no-code framework based on ns-3, in: Proceedings of the 2022 Latin America Networking Conference, LANC '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 42–49, http://dx.doi.org/10.1145/3545250.3560845.

[37] E. Black, S. Gamboa, R. Rouil, NetSimulyzer: A 3D network simulation analyzer for ns-3, in: WNS3 '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 65–72, http://dx.doi.org/10.1145/3460797.3460806.

[38] G.F. Riley, J. Abraham, NetAnim, 2023, https://www.nsnam.org/wiki/NetAnim, (Accessed December. 15. 2023).

[39] G.J.A.M. Carneiro, Transparent Metropolitan Vehicular Network — Design and Fast Prototyping Methodology (Ph.D. thesis), Universidade do porto, Porto, 2012, Available at https://repositorio-aberto.up.pt/bitstream/10216/68161/1/000155388.pdf.

[40] H. Fontes, T. Cardoso, M. Ricardo, Improving ns-3 emulation performance for fast prototyping of network protocols, in: Proceedings of the 2016 Workshop on Ns-3, in: WNS3 '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 108–115, http://dx.doi.org/10.1145/2915371.2915374.

[41] H. Patel, H. Hiraskar, M.P. Tahiliani, Extending network emulation support in ns-3 using DPDK, in: Proceedings of the 2019 Workshop on Ns-3, in: WNS3 '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 17–24, http://dx.doi.org/10.1145/3321349.3321358.

[42] M. Jung, A. Hergenröder, OMNeTA: A hybrid simulator for a realistic evaluation of heterogeneous networks, in: Q2SWinet '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 75–82, http://dx.doi.org/10.1145/2815317.2815331.

[43] Linux-wpan: IEEE 802.15.4 & 6LoWPAN in Linux, 2024, https://github.com/linux-wpan/, (Accessed April. 23. 2024).

[44] D. Rakhmatov, S. Vrudhula, An analytical high-level battery model for use in energy management of portable electronic systems, in: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281), 2001, pp. 488–493, http://dx.doi.org/10.1109/ICCAD.2001.968687.

[45] A. Gallegos Ramonet, A. Guzman Urbina, K. Kinoshita, Evaluation and extension of ns-3 battery framework, in: WNS3 '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 102–108, http://dx.doi.org/10.1145/3592149.3592156.

[46] H. Wu, S. Nabar, R. Poovendran, An energy framework for the network simulator 3 (ns-3), in: 4th International ICST Conference on Simulation Tools and Techniques, 2012.

[47] N. Sastry, D. Wagner, Security considerations for IEEE 802.15.4 networks, WiSe '04, Association for Computing Machinery, New York, NY, USA, 2004, pp. 32–42, http://dx.doi.org/10.1145/1023646.1023654.

Alberto Gallegos Ramonet received his M.S. and PH.D. degrees in engineering from Ritsumeikan University in 2014 and 2018 respectively. From 2019 to 2021 he was Assistant Professor at the College of Information Science and Engineering at Ritsumeikan University. In 2021, he joined the graduate school of technology industrial and social sciences at Tokushima University where he currently holds the position of Assistant Professor. His research interests include but are not limited to wireless sensor networks, network simulations, routing protocols, MAC layer designs and Internet of things applications.

Tommaso Pecorella received Ph.D. and M.Sc. degrees in Electronic Engineering (Telecommunications track) from the Department of Information Engineering at University of Florence (Italy) in 2000 and 1996 respectively. From 2001 to 2007 he was a researcher at Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT). Since November 2007 he is a tenure track Assistant Professor in the Department of Information Engineering at University of Florence (Italy). In 2018 and 2019 he was also visiting professor at University of Saint Louis, Missouri (USA). He received the Best paper award at the IEEE GLOBECOM 2016, and in 2021 got the Italian Habilitation (Abilitazione Scientifica Nazionale) for Associate Professorship in Telecommunication Engineering. He is the author of more than 90 publications between conference papers and journals. His research interests focus on IoT communication systems, network security, and application of machine learning to networking systems.

Benedetta Picano (Member, IEEE) received the B.S. degree in computer science, the M.Sc. degree in computer engineering, and the Ph.D. degree in information engineering from the University of Florence, Florence, Italy. She was a Visiting Researcher with the University of Houston, Houston, TX, USA. Her research interests include matching theory, nonlinear time series analysis, digital twins, microservices, resource allocation in edge and fog computing infrastructures, and machine learning.

Kazuhiko Kinoshita received the B.E., M.E. and Ph.D degrees in information systems engineering from Osaka University, Osaka, Japan, in 1996, 1997 and 2003, respectively. From April 1998 to March 2002, he was an Assistant Professor at the Department of Information Systems Engineering, Graduate School of Engineering, Osaka University. From April 2002 to March 2008, he was an Assistant Professor at the Department of Information Networking, Graduate School of Information Science and Technology, Osaka University. From April 2008 to January 2015, he was an Associate Professor at the same University. Since February 2015, he has been a Professor at Tokushima University, Japan. His research interests include mobile networks, network management, and agent communications. Dr. Kinoshita is a senior member of IEEE and a fellow of IEICE.