# Unifying Radio-in-the-Loop Channel Emulation and Network Protocol Simulation to Improve Wireless Sensor Network Evaluation

Sebastian Böhm(✉) and Michael Kirsche

Computer Networks and Communication Systems Group,
Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany
{sebastian.boehm,michael.kirsche}@b-tu.de

**Abstract.** Evaluations of Internet of Things (IoT) and Wireless Sensor Network (WSN) applications demonstrate the significant and still existing gap between examinations with generic simulation environments and real-life (e.g., field test) or controlled (e.g., testbed) sensor network deployments in terms of realistic and accurate results. The separated use of single examination approaches is often not enough to overcome all evaluation challenges. We therefore propose a combination of discrete-event simulation, radio-channel emulation, and real hardware working together on different layers of the protocol stack of the system-under-test. Our combined approach reduces the gap between abstract simulations and network testbed experiments by providing adjustable radio conditions for repeatable evaluations of WSN and IoT networks.

## 1 Introduction

Practical applications of Wireless Sensor Networks (WSNs) require extensive testing and evaluation strategies that cover all layers of the protocol stack: from application and protocol data flows to radio channel influences and physical side-effects. These pre-deployment tests can be performed either via simulation, through emulation, or with the help of real-life testbeds. While a simulation of a networked system requires (abstract) models and representations of the system-under-test, emulations use components from the original system and try to emulate the usage scenarios. Real-life testbeds, in turn, try to replicate the exact application conditions prior to a rollout. Each of these network evaluation techniques has specific strengths and weaknesses when applied to WSN and IoT use cases. We discuss all three techniques in the next paragraphs and motivate our hybrid approach that combines simulative and emulative methods. Additional discussions of the evaluation methods are available in [1–4].

### WSN and IoT Simulation

*Simulation* allows one to easily construct layered network protocol architectures, device topologies, and algorithmic applications. Simulation model parameters

M. Baum et al. (Eds.): SimScience 2017, CCIS 889, pp. 219–238, 2018.
https://doi.org/10.1007/978-3-319-96271-9_14

can be customized to examine certain protocol behavior and performance issues. In the academic world, simulation is the de facto first step for the implementation of WSN solutions because no hardware is required to test new protocol designs. Common network simulators like *OMNeT++* [5] or *NS-3* [6] focus on the Discrete Event Simulation (DES) paradigm which models a system by its states. System state changes occur at discrete points in time. Examples of state changing events can be the start of a packet transmission or the expiration of a timer. A DES simulator jumps from one event to the next, skipping the time between events. Pseudo Random Number Generators (PRNGs) are used within the simulation process to randomize state changes and event occurrence. Simulator implementations use a Future Event Set (FES) and event routines (handlers) to create and schedule events. Algorithm 1 (adapted from [7]) shows the described general operations flow of discrete event scheduling in pseudo-code.

**Algorithm 1.** Discrete Event Scheduling

```
Precondition: Initialize simulation model and FES
while (FES not empty) and (simulation not complete) do
    fetch first event $e$ from FES
    advance simtime with event timestamp $t$
    Function process_event(e):
        perform model state transition
        if (create new event $v$) or (cancel event $w$) then
            insert $v$ into FES and delete $w$ from FES
        end
    return
end
```

The radio channel conditions and the operation environment characteristics have a strong influence on WSNs. These two aspects are typically simplified or completely omitted in network protocol simulations. Add-on frameworks that enable the modeling of wireless environments and channel conditions are discussed in [8, Sect. 2]. Their implementation, however, is based on mathematical functions with varying abstraction levels and complexity. Wireless channel modeling is a very complicated process (cp. [9]) with a significant tradeoff between the realism of simulation models and their performance and scalability [8, Sect. 4]. These drawbacks generally lead to a low confidence in WSN simulation results.

## WSN and IoT Testbeds

Application-specific *physical testbeds* for distributed sensor networks, on the other hand, are complex to set-up, to manage, and to operate. They are also cost intensive in comparison to other evaluation approaches. [10] summarizes and compares existing WSN experimentation testbeds and their characteristics. Regardless of the application scenario, testbed nodes operate in a radio environment which is often uncontrollable and may differ from the actual working environment of the system-under-test. Testbed evaluations enable an accurate representation of a sensor node's hardware (if available and installed in the testbed) and software. Monitoring and inspection of run-time characteristics is possible when firmware extensions and additional hardware are used to provide a feedback loop. However, such extensions may change the run-time execution or introduce unforeseen behavior and errors during the evaluation process.

**Radio Channel Emulation**

*Radio Channel Emulation* offers a controllable radio environment with constant Radio Frequency (RF) conditions to evaluate a system-under-test. WSN nodes are isolated from each other and connected over their radio interfaces to the RF channel emulator hardware, which emulates signal propagation effects. A common and important feature of these systems is the ability to control the large-scale fading between transmitters and receivers. Practical deployments vary from laboratory test setups with coaxial-based radio links to complex analog or digital radio channel emulators with support for hundreds of interconnected nodes. Nevertheless, applications run as static firmware implementations on real-life WSN hardware and flexible adjustments of applications or protocol parameters are complicated compared to network simulators.

The evaluation of WSN designs still offers many unsolved research challenges (cp. [3] for example), as the separated use of single approaches is often not enough to overcome all evaluation challenges (e.g., realistic cross-layer communication evaluation). To mitigate most of the flaws of the evaluation methods discussed in the previous three paragraphs we propose a combination of simulation, radio-channel emulation, and real hardware working together on different layers of the protocol stack of the system-under-test. The remainder of the paper describes the methodology of our hybrid approach in Sect. 2, gives a short overview of our background and related work in Sect. 3, presents a case study of our Hardware-in-the-Loop-based network emulation in Sect. 4, and discusses evaluation methods in Sect. 5 before we conclude the paper in Sect. 6.

## 2 Methodology

*Hardware-in-the-Loop (HIL)* evaluation concepts are well established for automotive, aerospace, and robotic application domains. In these areas, HIL approaches are used to verify existing hardware module implementations by triggering the hardware inputs via simulated events and observing the generated outputs and reactions of the hardware-under-test. In case of WSNs and wireless networked embedded systems, HIL concepts (e.g., [11,12]) for the evaluation of WSN applications and protocols are rarely implemented (cp. [13]). Depending on the development stage and the actual part of the system that is tested and fed with simulated data, name variations like model- or software-in-the-loop are widely-used. Our approach of combining radio channel emulation with a HIL concept for WSN testing is subsequently called Radio-in-the-Loop (RIL).

Figure 1 depicts a protocol stack of a sensor node and the three collaboration levels (numbered ① to ③). Time-discrete event simulations (level ① in Fig. 1) of the upper layers of the protocol stack (e.g., application, transport, and network layer) simplify the construction of different architectures and application models by using simulators like OMNeT++ and accompanied frameworks like INET[1].

Using real hardware (level ② in Fig. 1) to accurately represent parts of the Medium Access Control (MAC) sublayer and the complete Physical (PHY) layer mitigates a major drawback of typical WSN simulations. MAC and especially PHY simulation models are often abstracted, even though simulation frameworks might provide accurate models for upper layer protocols. WSN-specific simulators like Cooja [14] can provide more accurate representations of real WSN hardware due to real-life code execution, but their PHY and wireless propagation models are abstracted just like the ones from generic simulators like OMNeT++. Even more problematic is the restriction of WSN-specific simulators like Cooja to specific operating systems (i.e., Contiki [15] for Cooja). We combine real WSN hardware with upper layer network simulations over a Hardware-in-the-Loop approach to enable an accurate representation of the lower layers combined with the protocol and model variety of OMNeT++/INET.

We use RF channel emulation (level ③ in Fig. 1) to bypass the discussed drawbacks of WSN testbeds. By emulating the wireless channel in a controllable environment, we can provide adjustable PHY conditions comparable to simulation environments, while refraining from using abstracted wireless propagation and PHY models. Section 4 describes all three collaboration levels in detail. This Radio-in-the-Loop concept leverages on the strengths of the individual approaches to enable accurate and likewise flexible tests of WSNs.

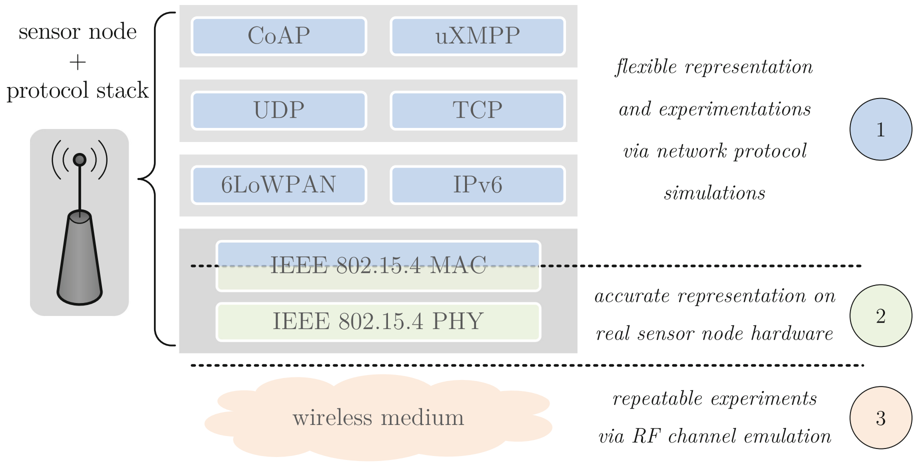
sensor node
+
protocol stack
CoAP
uXMPP
UDP
TCP
6LoWPAN
IPv6
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY
wireless medium
flexible representation and experimentations via network protocol simulations
1
accurate representation on real sensor node hardware
2
repeatable experiments via RF channel emulation
3

**Fig. 1.** Combining protocol simulation, real hardware, and RF channel emulation

[1] INET framework website: https://inet.omnetpp.org/.

The following three example scenarios benefit from using our RIL approach:

(i) The radio channel emulation (level ③ in Fig. 1) enables the definition of exceptional network topologies and various channel conditions that are difficult to reproduce within real world tests, while we can observe the behavior of the upper layer protocols on the simulated level.
(ii) With respect to the first scenario, cross-layer communication approaches and protocols that also involve radio medium access and physical data transmissions can benefit from accurate parameters and measurements of the transceiver chip hardware (level ② in Fig. 1) to get dependable evaluation results within a simulation-driven test setup.
(iii) Nevertheless, a small-scale setup of nodes in a controlled channel emulation environment can be extended easily with additional purely virtual nodes from collaboration level ① in Fig. 1 to increase the network traffic and simulate communication data stimuli for real wireless transmissions.

## 3 Related Work and Background Information

The majority of current simulation approaches [16] abstract heavily from real target hardware. Specialized simulators as discussed in [8,17] allow only a partial modeling of wireless environments and channel conditions. The underlying models are always based on abstracted mathematical functions with varying complexity. COOJA [14] provides an operation-system-specific simulation engine to enable software-based emulation of WSNs with a focus on simulating the code execution on the target hardware. COOJA supports three different models for wireless transmissions with varying parametrization: Unit Disk Graph Medium (UDGM), Direct Graph Radio Medium (DGRM), and Multi-path Ray-tracer Medium (MRM). Other simulation and emulation concepts for signal propagation or PHY layer support are surveyed, for example, in [8] and [16].

Coupling OMNeT++ with hardware is also considered in [18]. The authors describe a new HIL interface and changes in the OMNeT++ event scheduler to support a real-time exchange of messages over external hardware interfaces. [18] differs from our approach in terms of the underlying idea of joining radio channel emulation with simulation and in regard of the application scenario (i.e., home automation). The report describes an OMNeT++ gateway that basically forwards messages from one real-life device to another one via OMNeT++ without considering a radio channel emulation. Tests and evaluation of real-life hardware via simulated stimuli are also not yet considered in [18].

### 3.1 Network Protocol Simulation with OMNeT++

OMNeT++ is a popular open source DES simulator that is frequently used for communication network research. OMNeT++'s concept of exchanging messages via gates between modules (reusable building blocks) facilitates the development and simulation of complex scenarios. The INET add-on framework provides a multitude of simulation models for upper layer protocols.

In [19], we introduced a new simulation model for the popular IoT and WSN communication standard IEEE 802.15.4 [20]. The OMNeT++ model was created to simulate the complex behavior of the 802.15.4 MAC and PHY layers in a detailed fashion. We modeled the two layers with their connecting interfaces and the used service primitives according to the IEEE standard specifications and general modeling guidelines for 802.15.4 [21]. The structure of the OMNeT++ simulation model is depicted in Fig. 2.

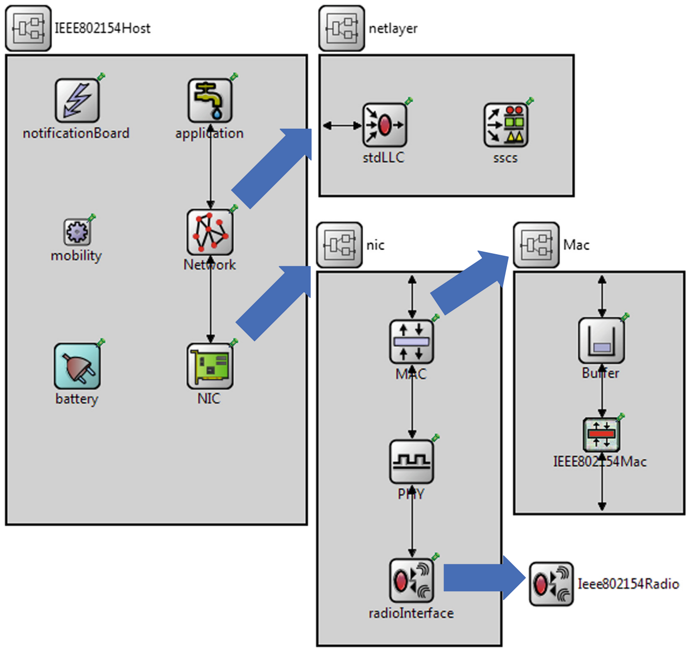

**Fig. 2.** Block diagram of the OMNeT++IEEE 802.15.4 model

The model itself consists of several layers and individual model components that are combined into a so-called *IEEE802154Host*. The *NIC* includes the parts of the IEEE standard that are most relevant for the communication. Listing 1.1 shows that the Protocol Data Unit (PDU) packet definitions include all types and fields that are specified in the IEEE 802.15.4 standard [20].

```
packet mpdu
{
    unsigned short fcs;          // 16-Bit Frame Check Sequence
    Ash ash;                     // Auxiliary Security Header
    // ... MAC frame payload is encapsulated ...
    MACAddressExt src;           // 0, 16 or 64-Bit Source Address
    unsigned short srcPANid;     // 0 or 16-Bits for Source PAN ID
    MACAddressExt dest;          // 0, 16 or 64-Bit Destination Address
    unsigned short destPANid;    // 0 or 16-Bits for Destination PAN ID
    unsigned char sqnr=0;        // 8-Bit Sequence number
    unsigned short fcf=0;        // 16-Bit Frame Control Field
}
```

**Listing 1.1.** MPDU packet definition - excerpt from `MPDU.msg`

### 3.2 RF Channel Emulation with the RoSeNet Testbed

In addition to OMNeT++, we work with RoSeNet[2], a network emulation platform for low-power wireless technologies that focuses on hardware-based channel emulation via a controllable coaxial cable radio environment (level ③ in Fig. 1). The modular system incorporates interconnected *emulation panels* that manage multiple sensor nodes on designated slots. By adjusting the signal attenuation values among nodes and panels in this shielded RF environment, it is possible to emulate distances or geographical positions and topologies of networked nodes. At designated signal supply points, interference signals can also be injected into the signal path. The overall architecture enables the emulation of large-scale networks with up to 1000 wireless sensor nodes (Fig. 3).

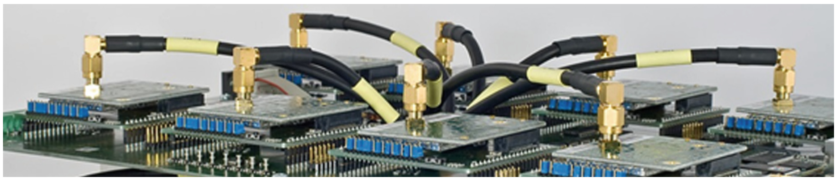

**Fig. 3.** RoSeNet emulation and test platform (taken from RoSeNet web page)

For our first HIL coupling experiments [22], we developed interfaces to transmit generated MAC layer frames via real sensor node hardware to achieve control over the communication flow in the network. In order to generate traffic for an initial test of the HIL system we simply inject generated protocol data frames. We added message handlers for the *emulation control server* application that are responsible to start the packet transmission to the hardware. The *Management Controller* on the addressed hardware panel forwards this packet to the designated slot of the transmitter node. A detailed description of the frame transmission and reception at node level is given in Sect. 4.3.

## 4 An OMNeT++ and RoSeNet RIL Architecture

Our current prototype includes extensions for OMNeT++/INET and the IEEE 802.15.4 simulation model (cp. Sect. 3.1), the hardware interfaces on the RoSeNet emulation testbed, and a *Forwarder* implementation that acts as bridge between the two domains. Figure 4 illustrates the basic Radio-in-the-Loop setup and the abstracted message exchange among the collaborating entities.

[2] RoSeNet radio channel emulation platform: https://www.dresden-elektronik.de/ingenieurtechnik/development/research/rosenet/.

For data exchange, we use the Packet Capture (PCAP) file format, the de facto standard capture format for network packet traffic. With the *PCAP Next Generation* (PCAPNG)[3] extension, we can exchange additional information for data packets, for example the interface identifier for multiple external devices.

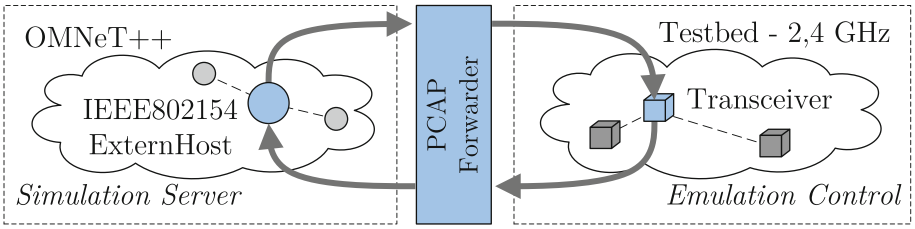

**Fig. 4.** Simplified message flow between OMNeT++ and testbed hardware

## 4.1 RIL Simulator Interfaces

We extended the IEEE 802.15.4 simulation model with a new module, called `IEEE802154ExtHost`, that enables the RIL operation. In the current implementation, an *external host* sends simulated MAC frames to an *external interface* instead of the PHY model. Figure 5 shows that the `IEEE802154ExtInterface` operates with a `IEEE802154Serializer` and a `PCAPScheduler` to convert between raw PCAP data bytes and OMNeT++MAC frame objects.

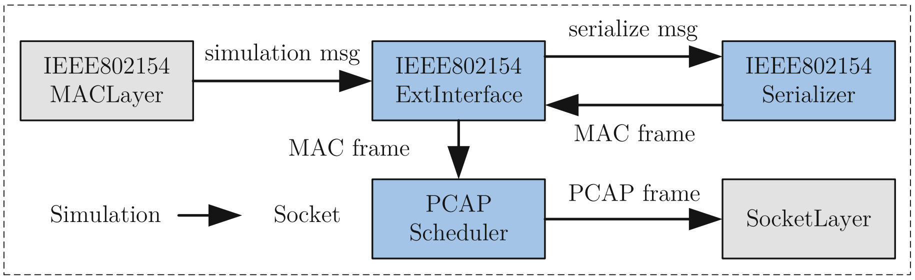

**Fig. 5.** IEEE 802.15.4 external interface in OMNeT++/INET

`PCAPScheduler` is derived from OMNeT++'s `cRealTimeScheduler` class. In OMNeT++, the event scheduler is one of the most important components, as it controls the event processing and manipulates the Future Event Set (cp. Sect. 1). The scheduler class provides a function, called `setInterfaceModule()`, which enables the connection of external interfaces (i.e., a Transmission Control Protocol (TCP) socket in our case) to the simulation. The scheduler function

[3] PCAPNG capture file format: https://github.com/pcapng/pcapng.

`getNextEvent()` is synchronized to the real-time clock and checks periodically if an event occurred at the socket. With respect to the PCAPNG file format, the scheduler implements various functions for writing and especially reading the specified blocks from the socket stream in the separate `PCAPNGReader` module. For the handling of PCAPNG, we only implemented the three basic block types that are relevant to our use case. These are:

| | | |
|---|---|---|
| `handleSHB()`: | Section Header Block (SHB) | *(init PCAP handling)* |
| `handleIDB()`: | Interface Description Block (IDB) | *(set hardware interface)* |
| `handleEPB()`: | Enhanced Packet Block (EPB) | *(process MAC packet)* |

The `IEEE802154ExtInterface` cooperates with the `IEEE802154Serializer` module to convert between simulation and real-life packet formats. The *external interface* is also responsible for handling all incoming and outgoing packet data traffic for the *scheduler*. A function named `handleMessage()` deals with events that can be either Radio-in-the-Loop messages or regular simulation events. Incoming *external* 802.15.4 MAC frames are deserialized from the serializer and directly send to the corresponding simulation node module.

The mapping between simulation nodes and PCAPNG data is performed with the help of an interface table that stores the simulation module identifier for the corresponding hardware identifier.

## 4.2 PACP Forwarder

A transparent forwarder application interchanges data in the PCAPNG format between the simulator and the target emulation system. The implementation uses threads and acts as a dispatcher and aggregator of PCAPNG data streams. On the hardware side, assigned destinations can be single sensor node platforms, individual transceiver chips, or whole testbed control systems. While PCAPNG is able to handle multiple hardware interfaces and link layer protocols, the forwarder application can aggregate data from different end-devices to constitute a single socket data stream for further scheduling in the OMNeT++ simulation. Figure 6 depicts the main PCAPNG block types and the packet processing.

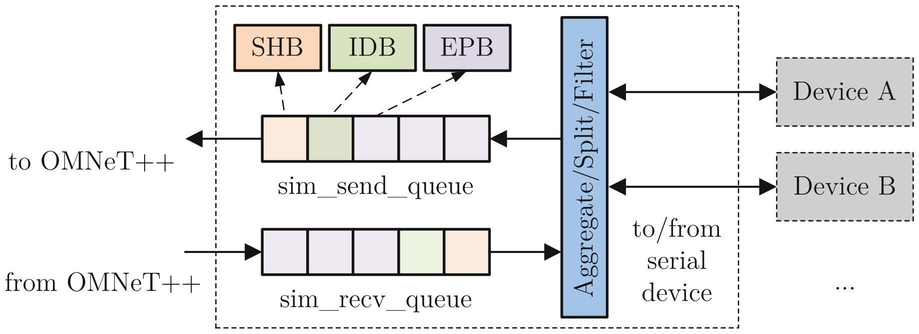

**Fig. 6.** PCAP forwarder architecture

For our purposes, we use the PCAPNG block types SHB, IDB, and EPB. The Section Header Block (SHB) and one Interface Description Block (IDB) for every used hardware node are exchanged at the beginning of the scenario execution. The Enhanced Packet Blocks (EPBs) represent the MAC data frames.

## 4.3 Node Simulation/Emulation Firmware

Contiki OS [15] is used to enable the reception of PCAP protocol frames via a Universal Asynchronous serial Receiver and Transmitter (UART) interface to create the respective MAC events and to transmit real radio frames among sensor nodes on the introduced RoSeNet emulation platform. We currently use nodes with ATmega128RFA1 radios[4], but the Transceiver (TRX) firmware can also be implemented for an arbitrary node platform and other WSN operating systems. Algorithm 2 illustrates the transceiver process with the corresponding transmit and receive functions in pseudo-code, derived from our current Contiki-based implementation. The required *promiscuous mode* in Algorithm 2 enables a Network Interface Controller (NIC) to pass all received network traffic captured from the medium to the system's Central Processing Unit (CPU).

**Algorithm 2.** Nodes Transceiver Process

```
Precondition: Initialize radio driver in promiscuous mode
Function receiver_callback()                          ▷ called by radio driver
    record timestamp and create PCAP frame
    send out PCAP frame via serial interface
return
while (true) do
    wait until PCAP serial interface event occur
    copy MAC frame into the packetbuf
    Function sender_callback()
        send out frame via radio interface
    return
end
```

Our implementation is actually using a fixed transmission channel with a permanently activated RF transceiver. The Carrier Sense Multiple Access Collision Avoidance (CSMA-CA) protocol is used to control and regulate the channel access. A *pcap_line* input process state machine creates PCAP events from frames received over the UART interface. A *sender_callback* function immediately transmits the frame onto the wireless channel. At the moment, we are able to transmit

[4] ATmega128RFA1: http://www.microchip.com/wwwproducts/en/ATmega128RFA1.

frames in both directions between OMNeT++/INET and common RS232 UART interfaces of typical IEEE 802.15.4 transceiver chips (e.g., ATmega128RFA1) and sensor nodes on the introduced RoSeNet emulation system.

## 4.4 Radio Emulation

The setup of the emulation scenario and the necessary parameters follow the typical OMNeT++ guidelines for the creation of simulation scenarios. We create `.ned` and `.ini` files for OMNeT++ that include the definition of the network topology as well as necessary parameters and simulation options. By using RoSeNet's radio emulation architecture, we are able to emulate a long term fading of signal transmissions. With reference to the Free Space Path Loss (FSPL) (cp. [9, Sect. 2]), we can *arrange* testbed nodes in virtual geographic positions. We take two-dimensional coordinates of the modeled scenario and the RF parameters of the sensor node hardware to determine the signal fading between individual wireless sensor nodes. We compute the signal loss between nodes (FSPL) with a simplified uniform spread of energy in free space given by the path loss model in Eq. 1, adapted from [9, Sect. 2.5].

$$FSPL(dB) = -20\log_{10}\left(\frac{4\pi d}{\lambda}\right) \tag{1}$$

$$\begin{aligned} \lambda &- \text{ transmission channel center frequency wavelength} \\ d &- \text{ the distance between sender and receiver} \end{aligned}$$

One important constraint is that RoSeNet has both fixed (i.e., integrated into the platform, non-adjustable) and variable signal attenuators (i.e., adjustable by the user) in its coxial environment. The quintessence and problem at the same time is to allocate nodes and set all involved attenuators to their according values to achieve the desired radio topology. For our RIL scenarios, we therefore need to create radio topologies that are representable in the two domains.

### Hardware Allocation on the Emulation Testbed

A RIL testbed architecture with coaxial-based radio links can be represented as an undirected communication graph $G$. With RoSeNet, we have a plain tree structure (cp. Fig. 7) in which the root is the central *anchor node* connected to *chains*. One chain has several modular entities called *panels*, which include the actual sensor nodes (the tree leaves), as it was introduced in Sect. 3.2.

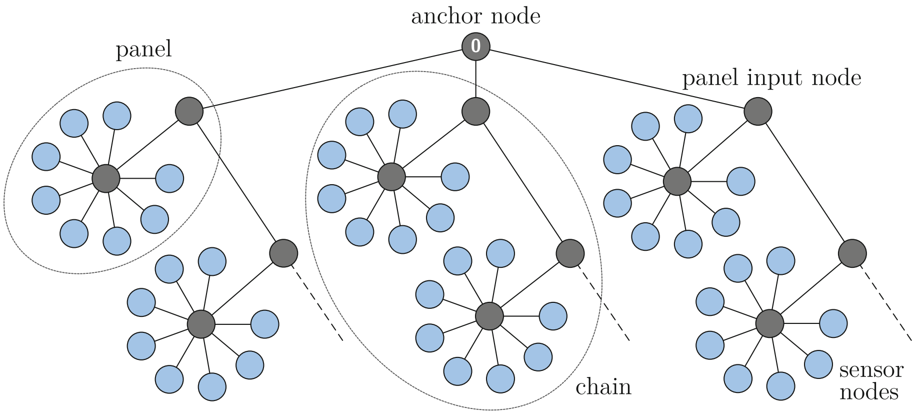

**Fig. 7.** RoSeNet's emulation architecture as an undirected communication tree $G_{A_T}$

We modeled the graph-based abstraction and several RF dependencies of the target emulation hardware for our allocation scheme. An overview of a number of important variables of the model definition is given below.

| | | |
|---|---|---|
| $N$: | a set of WSN hardware node types | $\{'RCB128RFA1', \ldots\}$ |
| $H$: | a set of emulator RF node types | $\{'anchor','splitter','input'\}$ |
| $A$: | a set of static attenuation values | $\{1, 3, 11, 21, 31\} \subseteq \mathbb{N}$ |
| $I_\lambda$: | a set of graph $G$ invariants | |
| $\epsilon$: | deviation of attenuation values | $\{\epsilon \in \mathbb{R}_+ \mid \epsilon < 1\}$ |
| $a_e$: | attenuation value of a graph $G$ edge | $e \in E(G_S)$, $a_e \in \mathbb{N}$ |
| $G$: | an undirected communication graph | $G = (V, E)$ |
| | a set of nodes from $G$ | $V = \{v \mid v \in N\}$ |
| | a set of weighted edges from $G$ | $E \subseteq \{(i, j, a) \mid i, j \in V; a \in \mathbb{N}\}$ |
| $G_A$: | an undirected allocation graph | $G_A = G = (V, E)$ |
| | a set of nodes from $G_A$ | $V = \{v \mid v \in \{N \cup H\}\}$ |
| $G_{A_T}$: | an undirected allocation tree | $G_{A_T} \subseteq G_A = G = (V, E)$ |
| | a set of nodes from $G_{A_T}$ | $V = \{v \mid v_r = anchor, v_l \in N\}$ |
| $G_H$: | an emulation hardware graph | $G_S = G_{A_T}$ |
| $G_S$: | a scenario graph | $G_S = G = (V, E)$ |

We designed a multiple stage allocation process to be able to automatically find attenuation values of the RF signal path on testbed platforms like RoSeNet. We use graph analysis and Mixed Integer Linear Programming (MILP) based on graph representations for the scenario and the hardware platform for this. The abstracted allocation procedure is given in Algorithm 3.

**Algorithm 3.** Node Hardware Allocation

```
Precondition: |scenario nodes| $\leq$ |hardware nodes|
Postcondition: $G_A \subseteq G_H$ with tolerance to $a_e$
generate $G_S$ and $G_H$
$I_\lambda$ = analyze scenario $G_S$                    ▷ $I_\lambda$ not finally specified
while (true) do
    $G_A$ = allocate nodes from $G_H$ with $G_S$ and $I_\lambda$
    Function calculate_attenuation($G_A$, $G_S$)
        ▷ create the Linear Program and add constraints
        make LP from $|V(G_S)|$ and $|V(G_A)|$
        add LP scenario constraints from $G_S$
        ▷ allow deviation from fixed values
        for $\epsilon \in Set_\epsilon$ do
            add LP hardware constraints from $G_H$ with $\epsilon$
            solve LP
            if attenuation found then
                allocate graph $G_A$
                return
            end
        end
    return
end
```

The allocation process includes multiple steps, starting with the generation of the scenario graph $G_S$ and the emulation hardware graph $G_H$ from the currently available hardware installation. In the second step, several parameters (graph invariants $I_\lambda$) of the scenario graph are calculated and processed by an initial node allocator, which makes a first decision regarding the panel-node placement $G_A$. This step of the process is only statically implemented for now. In the third step, we transfer our graph representation with the allocated nodes and all additional constraints (coming from the hardware and the RF dependencies) into a Linear Program (LP) which can then be solved using MILP. If a solution is calculable then we are done, otherwise we have to adjust our parameters (e.g., the deviation of the fixed attenuation parameters $\epsilon$) or we need to move back to step two to calculate a new panel-node placement $G_A$. If no solution can be calculated at all (e.g., due to hardware constraints), we have to change the initial radio topology or inform the user of the unsupported scenario.

## 5 Evaluation

The preceding sections gave an overview of our methodology and introduced the different submodules of our hybrid RIL approach. As the implementation and testing of the OMNeT++ and RoSeNet RIL architecture are still ongoing work, we will discuss evaluation approaches for submodules and different aspects of our approach and their feasibility to verify the usefulness of the OMNeT++/RoSeNet coupling for combined channel emulation and protocol simulation. Furthermore we present first results and fundamental steps of our submodule evaluation. A performance evaluation of the RIL simulator interfaces is discussed in Sect. 5.1. The PCAP Forwarder component is functionally evaluated to assure its correct behavior in Sect. 5.2. Section 5.3 shortly discusses a rudimentary performance evaluation of the emulation firmware prototype while Sect. 5.4 rounds off the evaluation part with an analysis of the hardware allocation and the MILP solver.

### 5.1 RIL Simulator Interfaces

An important aspect of real time simulations is the performance of the simulation implementation. When designing a simulation model, its implementation is usually less efficient when compared to the implementation of the real system. On the other hand, simulations typically run on high performance hosts. This difference is especially significant for WSN and IoT simulations of resource-constrained devices. Since we interconnect simulations and real hardware in our RIL approach, we have to ensure that the simulator maintains real-time capabilities for accessing the radio transceiver hardware.

Exemplary research like [23] already demonstrated sufficient throughput measurements of similar external interfaces for OMNeT++ in the past (as a result of their examples they fully utilize a link of 10 Mbit/s). When compared to the maximum throughput of IEEE 802.15.4 radio transceiver hardware (with a maximum over-the-air data rate of 250 kbit/s for the popular 2.45 GHz frequency band), we consider large scale test setups with dozens of sensor nodes to discover the limitations of our approach, for example in terms of the maximum number of RIL nodes. These results can finally be compared to the maximum theoretical throughput on a link.

As a fundamental evaluation step, we modified simulation modules to generate all frame types specified in the IEEE 802.15.4 standard [20, Sect. 7.2] and send them via our interface implementation. A test data receiver connects to the OMNeT++ socket at the local host and writes all received traffic into a single PCAP trace file. We use Wireshark[5], the de facto standard network protocol analyzer, to verify the correctness of the frame transmission.

### 5.2 PCAP Forwarder

Our forwarder application acts as a transparent bridge between the simulation and the emulation domain. We thus have to ensure the fast reception, processing, and transmission of frames. Depending of the exact scenario, the forwarder application can be evaluated by performance measurements when aggregating, splitting, and filtering PCAP data streams. For running corresponding performance measurements we need to define these concrete scenarios and generate suitable PCAP traces or schedule test runs together with a simulation run. We

[5] Wireshark network protocol analyzer: https://www.wireshark.org/.

consider precise time-stamping of processed PCAP frames with the help of high resolution clock timers to record the arrival and departure time for each frame.

We started by stress testing the application and measuring the overall data throughput by passing PCAP trace files from a sender socket via our forwarder to a receiver socket and vise versa (the receiver socket acts as a TCP repeater). Figure 8 shows the corresponding test setup (the dashed arrows show virtual connections; other connections represent the real data flow). We did not observe any dropped frames; all packets sent by the PCAP traffic generator were received by it after they passed through the PCAP forwarder and were returned by the TCP repeater. The test proves the correct functionality of the PCAP forwarder.

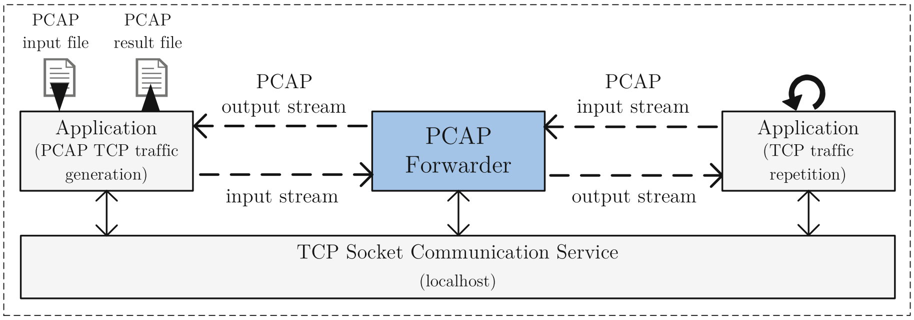
PCAP input file
PCAP result file
PCAP output stream
PCAP input stream
Application (PCAP TCP traffic generation)
PCAP Forwarder
Application (TCP traffic repetition)
input stream
output stream
TCP Socket Communication Service (localhost)

**Fig. 8.** Setup of the PCAP Forwarder application test

Looking at a data transfer of maximum sized IEEE 802.15.4 data frames (10k data frames with 127 bytes each – overall data amount of 1.52 MB), we measured an exemplary throughput of 91.5 Mbit/s from the sender (TCP traffic generation) to the PCAP Forwarder and 91.4 Mbit/s from the sender to the receiver module (TCP traffic repetition) on an Ubuntu Linux (64 bit) virtual machine with 7 vCPUs (Intel Xeon X3470 Quad Core @ 2.93 GHz). While traffic at the localhost gets processed by a loopback adapter in the kernel we cannot evaluate the precise throughput readings, but our first measurements show a 0.11% performance decrease caused by the forwarder. We assume that our packet handling routines, currently without considering aggregating, splitting, and filtering frames, do not have a significant influence on the overall data throughput.

## 5.3 Node Simulation/Emulation Firmware

For our emulation firmware prototype, we started with measurements of the maximum over-the-air transmission rates for different frame lengths for the currently used hardware platform RCB128RFA1[6]. We used the Texas Instruments

[6] RCB128RFA1 Radio Controller Board: http://www.dresden-elektronik.de/funktechnik/products/reference-designs/atmel-radio-controller-boards/radio-controller-boards/.

(TI) CC2531[7] transceiver module with TI's own SmartRF Packet Sniffer[8] software. We calculated the theoretical maximum transmission rate for frame transmissions between the serial and the RF interface, based on the used PHY specification and the microcontroller specs of our test hardware. Figure 9 depicts the results of the frame processing compared to the theoretical limitations. We achieve an overall throughput (independent of the frame size) of approximately >90% of the theoretical maximum packet rate at the serial interface. This performance decrease should primarily be caused by the PCAP-handling at the customized serial input driver and the frame type classification as well as the buffering of air-frames at the radio interface.

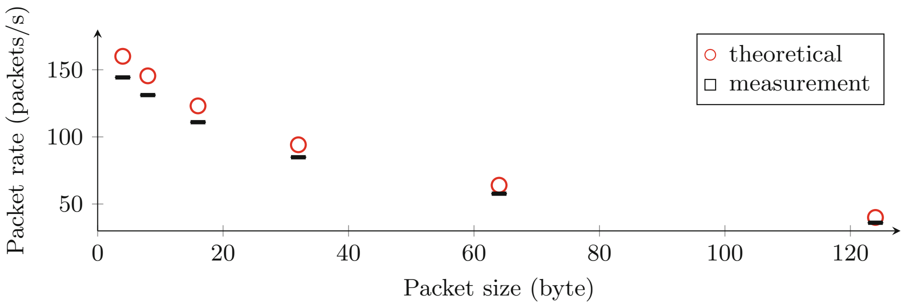

**Fig. 9.** Throughput of the PCAP firmware implementation

## 5.4 Hardware Allocation

The allocation process can be evaluated by measuring the algorithm run time in dependance of the scenario parameters (especially the number of nodes and the considered network topology). First of all, comprehensive and allocatable test scenarios need to be designed again. For the current panel hardware architecture the corresponding MILP model incorporates 378 constraints modeled by a total of 756 variables. Initial measurements of the allocation algorithm show a disproportionate increase in the iterations and calculation time with the number of scenario constraints to get optimal solutions from the MILP solver (see Fig. 10 – optimal). For the first steps of the allocation procedure we are not interested in optimal solutions but only in whether a model is feasible or not. To find a feasible solution for small scale test setups the solver in our simplified test cases needs less than 1000 iterations (see Fig. 10 – feasible). While running on an Ubuntu Linux (64 bit) virtual machine with 7 vCPUs (Intel Xeon X3470 Quad Core @ 2.93 GHz), the solver is able to calculate a solution whether a model is feasible or not in less than 0.5 s.

[7] Texas Instruments (TI) CC2531: http://www.ti.com/product/CC2531.
[8] SmartRF Protocol Packet Sniffer http://www.ti.com/tool/PACKET-SNIFFER.

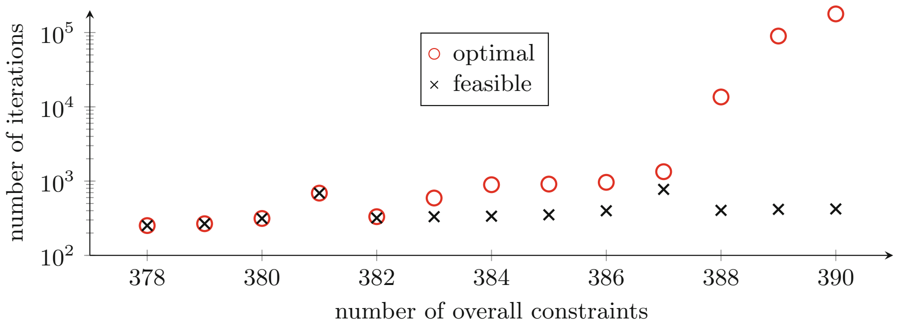

**Fig. 10.** Number of iterations for solving the allocation model

An overall evaluation of the system depends, in particular, on well defined emulation scenarios besides the interaction between all components. Furthermore, evaluation systems should always be additionally evaluated in an application-specific manner to determine which application types and which specific device topologies are meaningful to evaluate in general.

## 6 Summary and Outlook

WSN usage is spreading into all kinds of application domains over the last years. Applications and protocol stacks get more complex and thus more difficult to test and to evaluate. There is a growing demand in the simulation community to include hardware and RF environment-related details into network protocol simulation to facilitate the validation of simulative investigations and increase the realism of the abstracted representations of wireless channel characteristics.

In this work, we presented an approach to extend pure WSN protocol simulation through a Radio-in-the-Loop concept. We apply radio channel emulation with real sensor node hardware on the PHY layer to constitute a more precise behavior of WSN use cases. We focus on OMNeT++ and RoSeNet in our prototypical implementations and tests. However, the proposed ideas are not limited to a specific emulation system, hardware type, or network simulation environment. The support for wireless channel emulation using real hardware within network simulations will help to improve examinations of the interactions of protocol data flow and, for example, node energy consumption in reproducible radio conditions. Ultimately, the objective of our approach is to introduce a new tool chain for performance evaluation and cross-layer optimization in WSNs. This requires further evaluations and improvements which we discuss next.

In addition to pure MAC frame transmissions, the 802.15.4 simulation model needs to control the RF parameters of hardware-based radio transmissions, for example: the data rate, the modulation, the transmission power, the frequency, and the transmission channel. We use a TRX firmware (cp. Sect. 4.3) that sets these RF parameters inside the source code at compile-time for initial testing. To increase the overall model accuracy we plan to adjust the TRX firmware to enable the processing of IEEE 802.15.4 PHY data and service primitives which are already implemented in our simulation model. A more modular attempt could be to exchange those parameters in optional extension headers of the PCAPNG file format or in so-called *Radiotap*[9] headers (like they exist for IEEE 802.11 WLAN) and implement *setter* and *getter* methods for these parameters at the target hardware. This would allow the simulation to execute all procedures and features defined in the IEEE 802.15.4 standard and provide valid RF measurements for the simulated MAC layer.

Since most of the commercially available IEEE 802.15.4 transceivers only implement a subset of the complete PHY specification in practice, we also consider the use of flexible and reconfigurable Software Defined Radio (SDR) modules as RIL gateways instead of standard-conform but functionally limited transceivers. This approach could also help to avoid the data rate bottleneck of the UART interface of the transceiver modules. This architectural change opens up interesting possibilities for rapid prototyping of cross-layer communication approaches for future transceiver chip design.

Emulation of node mobility is another practical use case for test setups. In order to model mobility or node movement patterns (e.g., a receiver node moves out of the reception range of the transmitter node) within the RIL channel emulation, we have to vary hardware attenuation values during the emulation run-time. There are still open questions regarding the limits of the given RoSeNet system architecture for the emulation of mobility due to the complexity and the radio topology dependency of the hardware allocation process as it was discussed in Sect. 4.4.

**Acknowledgement.** Parts of this work were funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) through the Central Innovation Programme (ZIM initiative) under Contract No. ZF4119201ED5.

## References

1. Imran, M., Said, A., Hasbullah, H.: A survey of simulators, emulators and testbeds for wireless sensor networks. In: International Symposium in Information Technology (ITSim). IEEE (2010). https://doi.org/10.1109/ITSIM.2010.5561571
2. Kropff, M., Krop, T., Hollick, M., Mogre, P.S., Steinmetz, R.: A survey on real world and emulation testbeds for mobile ad hoc networks. In: Proceedings of the 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM), 6-pp. IEEE (2006). https://doi.org/10.1109/tridnt.2006.1649182
3. Göktürk, E.: Emulating ad hoc networks: differences from simulations and emulation specific problems. In: Tugcu, T., Gelenbe, E., Caglayan, M.U. (eds.) New Trends in Computer Networks. Advances in Computer Science and Engineering: Reports, vol. 1. Imperial College Press, October 2005
4. Wehrle, K., Güneş, M., Gross, J.: Modeling and Tools for Network Simulation, 1st edn. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3

[9] Radiotap project website: https://www.radiotap.org/.

5. Varga, A., Hornig, R.: An overview of the OMNeT++ simulation environment. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools), ICST, article no. 60 (2008)
6. Riley, G.F., Henderson, T.R.: The NS-3 network simulator. [4] Chapter 2, pp. 15–34. https://doi.org/10.1007/978-3-642-12331-3
7. OMNeT++ Manual: Function called for each event (2017). https://omnetpp.org/doc/omnetpp/manual/#sec:simple-modules:handlemessage:overview
8. Stehlik, M.: Comparison of simulators for wireless sensor networks. M.Sc. thesis, Faculty of Informatics, Masaryk University (2011)
9. Goldsmith, A.: Wireless Communications. Cambridge University Press, Cambridge (2005)
10. Dwivedi, A.K., Vyas, O.P.: An exploratory study of experimental tools for wireless sensor networks. Wirel. Sens. Netw. **3**(7), 215–240 (2011)
11. Duan, S., Wan, Y., Meng, P., Wang, Q.: Hardware-in-the-loop and parallel simulation architecture for WSN. TELKOMNIKA **11**(1), 103–114 (2013)
12. Mozumdar, M.M.R., Lavagno, L., Vanzago, L., Sangiovanni-Vincentelli, A.L.: HILAC: a framework for hardware in the loop simulation and multi-platform automatic code generation of WSN applications. In: International Symposium on Industrial Embedded Systems (SIES), pp. 88–97. IEEE (2010)
13. Papadopoulos, G.Z., Kritsis, K., Gallais, A., Chatzimisios, P., Noel, T.: Performance evaluation methods in ad hoc and wireless sensor networks: a literature study. IEEE Commun. Mag. **54**(1), 122–128 (2016)
14. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with COOJA. In: Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN), pp. 641–648. IEEE Computer Society (2006). https://doi.org/10.1109/LCN.2006.322172
15. Dunkels, A., Gronvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the 29th Annual IEEE Conference on Local Computer Networks (LCN), pp. 455–462. IEEE Computer Society (2004). https://doi.org/10.1109/LCN.2004.38
16. Sundani, H., Li, H., Devabhaktuni, V.K., Alam, M., Bhattacharya, P.: Wireless sensor network simulators a survey and comparisons. Int. J. Comput. Netw. (IJCN) **2**(5), 249–265 (2011)
17. Du, W., Mieyeville, F., Navarro, D., O'Connor, I., Carrel, L.: Modeling and simulation of networked low-power embedded systems: a taxonomy. EURASIP J. Wirel. Commun. Netw. **2014**(1), 1–12 (2014)
18. Wehner, P., Göhringer, D.: Internet of Things simulation using OMNeT++ and hardware in the loop. In: Keramidas, G., Voros, N., Hübner, M. (eds.) Components and Services for IoT Platforms, pp. 77–87. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-42304-3_4
19. Kirsche, M., Schnurbusch, M.: A new IEEE 802.15.4 simulation model for OMNeT++/INET. In: Proceedings of the 1st International OMNeT++ Community Summit (OMNeT 2014), September 2014
20. IEEE Standards Association: Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). IEEE Standards Document - Revision of IEEE Std. 802.15.4$^{\text{TM}}$-2006. IEEE, September 2006. https://doi.org/10.1109/IEEESTD.2006.232110
21. Kirsche, M.: Selected System Models - IEEE 802.15.4. [4] Chapter 12.3, pp. 276–303. https://doi.org/10.1007/978-3-642-12331-3

22. Böhm, S., Kirsche, M.: Looking into hardware-in-the-loop coupling of OMNeT++ and RoSeNet. In: Proceedings of the 2nd International OMNeT++ Community Summit (OMNeT 2015), September 2015
23. Tüxen, M., Rüngeler, I., Rathgeb, E.P.: Interface connecting the INET simulation framework with the real world. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques (SIMUTools 2008), ICST, Brussels, Belgium, Belgium, pp. 40:1–40:6. ICST (2008). https://doi.org/10.1145/1416222.1416267