Split-Protocol-Stack Network Simulation and Emulation of Wireless Embedded Systems with Radio-in-the-Loop

Towards Accurate Wireless Network Evaluation

Von der Fakultät 1 für MINT
– Mathematik, Informatik, Physik, Elektro- und Informationstechnik –
der Brandenburgischen Technischen Universität Cottbus–Senftenberg

genehmigte Dissertation

zur Erlangung des akademischen Grades eines

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

vorgelegt von

Master of Science

Sebastian Böhm

geboren am 09. Dezember 1987 in Guben

Vorsitzender:	Prof. Dr. rer. nat. habil. Klaus Meer
Gutachter:	Prof. Dr.-Ing. habil. Hartmut König
Gutachter:	Prof. Dr. rer. nat. Volker Turau

Tag der mündlichen Prüfung: 21. Oktober 2022

https://doi.org/10.26127/BTUOpen-6144

Abstract

Recent challenges and novel approaches of wireless communication networks are characterized by high performance requirements on the radio channel and by concepts of self-organization, in-network processing, or resource optimization which ultimately lead to distributed network applications, communication architectures, and radio transceivers. Associated evaluation is driven by a multitude of ever-increasing requirements that call for multidisciplinary expertise. Depending on the discipline, simulation is one of the most widely used technologies, whereby often abstract assumptions and models do not allow for sufficiently accurate, comparable results. In contrast, real-world measurements and field-tests can only be performed on actual systems, commonly under non-reproducible conditions.

This thesis establishes with the *Split-Protocol-Stack* a new type of evaluation method that intends to help closing the gap between purely simulative analyzes and real-world tests. With the inclusion of real radio hardware and radio channels in the event-based simulation, this central hybrid approach in connection with the *Radio-in-the-Loop* methodology creates synergies in interdisciplinary fields. The approach contains analytical discussions, methodological strategies, and practical contributions that are summarized as key elements in the subsequent central considerations and challenges.

With *Real-Time-Shift*, a pseudo-real-time synchronization approach for parallel simulation and radio channel emulation of communication flows is introduced. Based on the underlying time compensation scheme, the discrete event simulation is decoupled from real-time constraints when exchanging event messages with real-world wireless hardware. A physical layer emulation methodology and radio channel interface concept, called *Radio-in-the-Loop*, is introduced along with two practical realization approaches. Furthermore, strategic details on radio network planning with an approach to automatic hardware resource allocation are presented for radio channel emulation-capable network testbeds.

The contributions of this work are evaluated using real-world reference measurements, practical application scenarios, and experiments that provide proof of concepts. By means of an exemplary selected cross-layer optimization scenario, the benefits are practically demonstrated and discussed. Finally, based on IEEE 802.15.4 as the reference protocol standard for low-power wireless networks, this thesis provides feasibility studies and analysis results using the representative prototype *SEmulate* for the *Split-Protocol-Stack* approach.

Kurzfassung

Aktuelle Herausforderungen und neuartige Ansätze drahtloser Kommunikationsnetze sind geprägt durch hohe Leistungsanforderungen auf dem Funkkanal sowie Konzepte der Selbstorganisation, netzinternen Verarbeitung oder Ressourcen-Optimierung und münden letztendlich in verteilten Netzwerkanwendungen, Kommunikationsarchitekturen und Funk-Transceivern. Die Evaluation ist dabei stark von fachspezifischen, simulativen Analysen unter abstrahierten Annahmen und Modellen geprägt, so dass die Ergebnisse oft keine hinreichend genauen, vergleichbaren Schlussfolgerungen auf die Leistungsfähigkeit im Gesamtsystem zulassen. Praxistests hingegen, können nur für konkrete Systeme am Ende des Entwicklungszyklus unter zum Teil nicht reproduzierbaren Bedingungen durchgeführt werden.

Diese Arbeit begründet mit dem *Split-Protocol-Stack* eine neuartige Evaluationsmethodik die dazu beitragen soll, die Lücke zwischen rein simulativen Analysen und realen Tests zu schließen. Unter Einbeziehung realer Funkhardware und physikalisch realitätsnah nachgebildeter Funkkanäle in die ereignisbasierte Simulation schafft dieser zentrale hybride Ansatz in Verbindung mit der Methodik des *Radio-in-the-Loop* Synergien in interdisziplinären Fachgebieten. Der Ansatz enthält analytische Diskussionen, methodische Strategien und praktische Beiträge, die als Kernelemente in den nachfolgenden zentralen Betrachtungen und Herausforderungen zusammengefasst werden.

Unter der Bezeichnung *Real-Time-Shift* wird ein Synchronisationsansatz für die parallele Simulation und Funkkanalemulation von Kommunikationsabläufen eingeführt. Basierend auf der zugrundeliegenden Zeitkompensation wird die Simulation von Beschränkungen der Echtzeitfähigkeit beim Nachrichtenaustausch von Ereignissen mit reale Funk-Transceiver-Hardware entkoppelt. Darüberhinaus wird mit *Radio-in-the-Loop* eine Methode zur Emulation der physikalischen Schicht sowie Schnittstelle zum Funkkanal definiert und praktisch realisiert. Weiterhin werden strategische Details zur Funknetzplanung vorgestellt und mit einem Ansatz zur automatischen Hardware-Ressourcenzuweisung für die Funkkanalemulation analysiert und praktisch erprobt.

Bewertet werden die Beiträge dieser Arbeit durch Referenzmessungen, praktische Anwendungsszenarien und experimentelle Analysen, die den Nachweis der vorgestellten Konzepte erbringen sowie deren Nutzen evaluieren. Anhand eines exemplarisch ausgewählten Anwendungsszenarios der Protokollschicht-übergreifenden Optimierung, wird der Zugewinn praktisch demonstriert und diskutiert. Schließlich werden auf Basis von IEEE 802.15.4 als Referenz-Protokollstandard für drahtlose Niedrigenergie-Netze durch diese Arbeit Durchführbarkeitsstudien und Analyseergebnisse zusammen mit dem repräsentativen Prototypen *SEmulate* für den Ansatz des *Split-Protocol-Stack* bereitgestellt.

Table of Contents

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives and Contributions Overview . . . 8
- 1.3 Thesis Structure . . . 10

2 Wireless Network Simulation 11
- 2.1 Simulation-based Network Evaluation . . . 11
  - 2.1.1 Terminology and Classification of Simulation Modeling . . . 12
  - 2.1.2 Network Simulation Methodology . . . 14
  - 2.1.3 Discrete Event Simulation . . . 16
- 2.2 Modeling for Wireless Network Simulation . . . 19
  - 2.2.1 Radio Channel Modeling . . . 20
  - 2.2.2 Physical Layer Modeling . . . 22
  - 2.2.3 Data Link Layer Modeling . . . 25
  - 2.2.4 Higher Layer Modeling . . . 27
  - 2.2.5 Network Topology and Mobility Modeling . . . 29
- 2.3 Selected Simulation Systems and Models . . . 30
  - 2.3.1 Discrete-Event Simulators in the Wild . . . 30
  - 2.3.2 Link Layer Simulation Models and Systems . . . 31
- 2.4 Limitations of pure Simulations . . . 34
  - 2.4.1 Validation and Credibility . . . 34
  - 2.4.2 Performance and Scalability . . . 35
  - 2.4.3 Cross-Protocol Stack and Interference . . . 36

3 Wireless Network Emulation 37
- 3.1 Emulation Methodology and Classification . . . 37
  - 3.1.1 Methodical Delimitation . . . 38
  - 3.1.2 Classification . . . 38
- 3.2 Radio Link Emulation Requirements and Strategies . . . 40
  - 3.2.1 Emulator Requirements . . . 40
  - 3.2.2 Emulating Network Conditions . . . 42
  - 3.2.3 Emulating the Network Interfaces . . . 44

3.3 Selected Radio Link Emulation Approaches . . . 46
3.4 Limitations of Radio Link Emulation . . . 51
3.4.1 Cross-Layer and Cognitive Radio . . . 51
3.4.2 Real-Time Scalability and Mobility . . . 52

4 Parallel Simulation and Emulation 53
4.1 Methodology and Requirements . . . 53
4.1.1 Terminology and Methodical Delimitation . . . 54
4.1.2 Coupling Problems and Requirements . . . 55
4.2 Split-Protocol-Stack Wireless Network Emulation . . . 57
4.2.1 Pseudo-Real-Time Network Simulation . . . 58
4.2.2 Radio-in-the-Loop Wireless Transmissions . . . 60
4.2.3 Radio-in-the-Loop Analog Radio Channel Emulation . . . 61
4.3 Analysis of HIL and Co-Simulation Approaches . . . 62
4.3.1 Real Protocol Implementations in Simulated Networks . . . 63
4.3.2 Simulated Protocols on Real or Emulated Networks . . . 65
4.3.3 Analysis Summary and Concluding Discussion . . . 67
4.4 Benefits, Use Cases and Scenarios . . . 71
4.4.1 Approach Benefits Overview . . . 71
4.4.2 Application Domains and Use Cases . . . 72
4.4.3 Split-Protocol-Stack Reference Configuration Scenarios . . . 74

5 Real-Time-Shift Discrete Event Simulation & Synchronization 77
5.1 Real-Time Synchronization Problem Statement . . . 77
5.1.1 Real-Time Hardware-in-the-Loop Simulation . . . 78
5.1.2 Real-Time Synchronizing Simulation and Emulation . . . 79
5.2 Real-Time-Shift Network Simulation . . . 80
5.2.1 Event Transmission Latency and Jitter Compensation . . . 80
5.2.2 Pseudo-Real-Time Event Scheduling . . . 81
5.2.3 Real-Time Event Stream Processing . . . 84
5.2.4 Determination of the Time Constants . . . 86
5.3 Simulator Interfacing and Concept Implementation . . . 88
5.3.1 Interface Modeling and Abstraction . . . 88
5.3.2 Simulation Model Enhancements and External Interfacing . . . 89
5.3.3 Radio-in-the-Loop Event Stream Forwarder . . . 92
5.4 Evaluation and Discussion . . . 94
5.4.1 Emulation Scenario Model Validation and Verification . . . 94
5.4.2 Event-Stream Forwarder Performance and Accuracy . . . 96
5.4.3 Discussion of the Evaluation Results . . . 100

6 Radio-in-the-Loop Physical Layer Modeling 103
6.1 Radio-in-the-Loop Modeling Strategy . . . 103
6.1.1 Interfacing and Scheduling Virtual Events on Real Hardware . . . 104
6.1.2 Physical Layer Modeling . . . 105
6.2 Chip Radio-in-the-Loop Wireless Transmissions . . . 106
6.2.1 Message Event Handling at the Node Interface . . . 106
6.2.2 Radio-in-the-Loop Real-Time Scheduling . . . 106
6.2.3 Handling and Dispatching Protocol Primitives . . . 107
6.3 Software Radio-in-the-Loop Wireless Transmissions . . . 108
6.3.1 Data Link Layer Interface Modeling . . . 109
6.3.2 Time-Aware Event Scheduling in Streaming Systems . . . 109
6.3.3 Software-Defined Radio Physical Layer Modeling . . . 110
6.4 Evaluation and Discussion . . . 113
6.4.1 Feasibility and Performance of the Chip Radio Transceiver . . . 113
6.4.2 Feasibility and Performance of the Software Radio Transceiver . . . 115
6.4.3 Discussion . . . 120

7 Radio-in-the-Loop Channel Emulation 123
7.1 Radio Channel Emulation Modeling Strategy . . . 123
7.1.1 Wireless Network Planning . . . 124
7.1.2 Radio Topology and Channel Emulation Modeling . . . 126
7.2 Chicken Creek Reference Measurements . . . 129
7.2.1 Measurement Setup and Node Placement . . . 130
7.2.2 Measurement Results and RIL Emulation Relevance . . . 131
7.3 Hardware Allocation in Network Emulation Testbeds . . . 132
7.3.1 Problem Statement and Model Definition . . . 132
7.3.2 Allocation Approach . . . 135
7.4 Evaluation and Discussion . . . 138
7.4.1 Feasibility of Hardware Allocation . . . 138
7.4.2 Enhancing the Hardware Attenuation . . . 139

8 SEmulate Prototype and Use Case Study 141
8.1 SEmulate Overview . . . 141
8.1.1 Emulation Prototype Architecture . . . 142
8.1.2 Creating, Building, and Running Emulations . . . 143
8.2 Case Study Scenario: Cross-Layer Optimization . . . 145
8.2.1 Motivation and Objective . . . 145
8.2.2 Modeling and Definition . . . 146
8.2.3 Results and Evaluation . . . 148
8.2.4 Discussion and Further Enhancements . . . 149

9 Conclusion **151**
9.1 Achievements and Contributions 151
9.2 Outlook 152

A Analysis and Discussion Details **157**
A.1 Analysis of Radio-in-the-Loop Solutions 157
A.2 Analysis of Radio Link Emulation Systems 158
A.3 Analysis of HIL and Co-Simulation Approaches 159

B OMNeT++ / INET Overview **161**
B.1 Simulation Architecture and Components 161
B.2 Creating, Building, and Running Simulations 162

C Implementation and Configuration Details **163**
C.1 OMNeT++ 163
C.1.1 Scenario Configuration 165
C.1.2 Cross-layer Optimization Scenario 166
C.2 SEmulate Backend 167
C.2.1 Radio-in-the-Loop Event Stream Forwarder 167
C.2.2 Radio-in-the-Loop Transceiver Implementation 168
C.2.3 Prototype Configuration 170

D Technologies, Protocols, and Standards **173**

E Acronyms **179**

Bibliography **183**

List of Figures

1.1 Layered protocol stack architecture and semantics 2
1.2 IEEE 802.15.4 and associated protocols 4
1.3 Structural outline of this thesis 10

2.1 Model and system classification 13
2.2 Network simulator abstraction 14
2.3 Requirements and effects in communication networks evaluation 15
2.4 System state changes over simulated time 16
2.5 Behavior of a real-time simulation 18
2.6 Modeling domains of wireless transmission systems 19
2.7 Block diagram of the *IEEE 802.15.4* model in *OMNeT++*/INET 33

3.1 Radio channel emulation of wireless transmission systems 43
3.2 Example PHY SDR Transmitter 45
3.3 Analysis of radio link emulation systems and approaches 48
3.4 RoSeNet radio emulation and test platform hardware panel 49
3.5 RoSeNet hardware architecture 50

4.1 Communication relations in the *RIL* methodology 55
4.2 The Split-Protocol-Stack evaluation architecture 57
4.3 Overview analysis of approaches in simulated networks 64
4.4 Overview analysis of hybrid approaches on real or emulated networks . . 66
4.5 Configuration for pure wireless link emulated networks 75
4.6 Configurations for hybrid gateway networks 76

5.1 Time compensation in real-time event execution 81
5.2 Pseudo-real-time concurrent event scheduling 83
5.3 Example IEEE 802.15.4 PD DataIndication SDU 88
5.4 IEEE 802.15.4 simulation modules in *OMNeT++*/INET 89
5.5 Example of a Event-Stream Forwarder setup 93
5.6 Simulation event message log of the validation scenario 94
5.7 PCAP trace of the scenario protocol sequences 95
5.8 Histogramm of the real-time jitter 97
5.9 The ESF accuracy as a function of event transmission parameters. 98
5.10 The ESF scheduling accuracy as a function of internal parameters. 99

6.1 Block diagram of a O-QPSK PHY SDR Transmitter 108

6.2 The SDR transceiver model in GNU Radio . . . 108
6.3 PHY model of the SDR transceiver in GNU Radio . . . 111
6.4 Received radio packets with SmartRF . . . 113
6.5 Chip RIL packet transmission performance . . . 114
6.6 Received radio packets with SmartRF . . . 115
6.7 Deviation of the mean received RSSI . . . 116
6.8 Waterfall plot of the channel hop from 15 to 16 . . . 116
6.9 Received signal strength and packet count . . . 117
6.10 Histogram of the GRC PCAP scheduling jitter . . . 118
6.11 The GRC PCAP scheduling jitter according to the Beacon frame size . . . 118
6.12 Software RIL packet transmission performance . . . 119
6.13 Comparison of the Chip and Software Radio-in-the-Loop Solution . . . 122

7.1 Example unit disk graph representation . . . 126
7.2 Example constellation diagram of adaptive channel equalization . . . 128
7.3 Chicken Creek ecosystem monitoring . . . 129
7.4 Reference coordinates for the RSSI measurements . . . 130
7.5 RSSI measurement results . . . 131
7.6 Example scenario graph representation . . . 133
7.7 RF network representation of a single emulation panel . . . 134
7.8 Matrix representation of the MILP model . . . 137
7.9 Allocation of the scenario . . . 137
7.10 Number of iterations for solving the allocation model . . . 138
7.11 Attenuation measurements . . . 140

8.1 SEmulate Prototype RIL Architecture . . . 142
8.2 Overview of building and running an emulation . . . 144
8.3 IEEE 802.15.4 cross-layer communication based on an external PIB . . . 147
8.4 RX gain optimization from LQI . . . 148

B.1 Overview of building and running a simulation in *OMNeT++* . . . 162

C.2 Prototype scenario configuration in SEmulate . . . 171

List of Tables

2.1 Common PHY transmitter and receiver parameters 24
2.2 Comparison of recommended network simulation and emulation systems 31

3.1 Classification of network emulation systems 39
3.2 Radio link emulation systems and approaches overview 46

4.1 HIL and co-simulation approaches in simulated networks 63
4.2 HIL and co-simulation approaches on real or emulated networks 65

7.1 Variable definition for the scenario radio topology 133
7.2 Variable definition for the panel hardware 134

A.1 Analysis of Radio-in-the-Loop solutions 157
A.2 Analysis of radio link emulation systems 158
A.3 Analysis of HIL and co-simulation approaches 159
A.4 Analysis of HIL and co-simulation approaches 160

C.1 Selected emulation hardware commands and descriptions 171

1 | Introduction

Aurea Mediocritas

> *"The technique of network emulation is intended to bridge the gap between simulation experiments and real-world testing. Emulation inherits reproducibility and control from simulation, but makes possible direct experiments with real components under test, thus increasing the realism of the results. In this respect, emulation can be seen as aurea mediocritas, the golden mean between simulation and real-world trials that trades off reproducibility and control for realism."* [1]

1.1 | Motivation

With the ubiquitous and rapid growth of new communication technologies, the wireless medium is irretrievably gaining importance for many processes in almost all fields. The design and development of wireless communication systems and networks in general and *Wireless Sensor Networks (WSNs)*, *Low-power and Lossy Networks (LLNs)*, or the wireless *Internet of Things (IoT)* in particular is becoming increasingly complex. Prototyping and evaluating large-scale wireless communication systems is challenging. In this thesis we consider such a system as network of resource-constrained devices. Various application domains have very precise physical requirements, e.g., in terms of bandwidth, latency, or packet loss ratio, and, on the other hand, extensive functional or algorithmic demands, e.g., self-organization, in-network-processing, and resource limitation. Their requirements necessitate many components, complex algorithms, and procedures implemented mainly by means of distributed network applications, communication protocols, protocol stacks, reconfigurable interfaces, and radio transceivers.

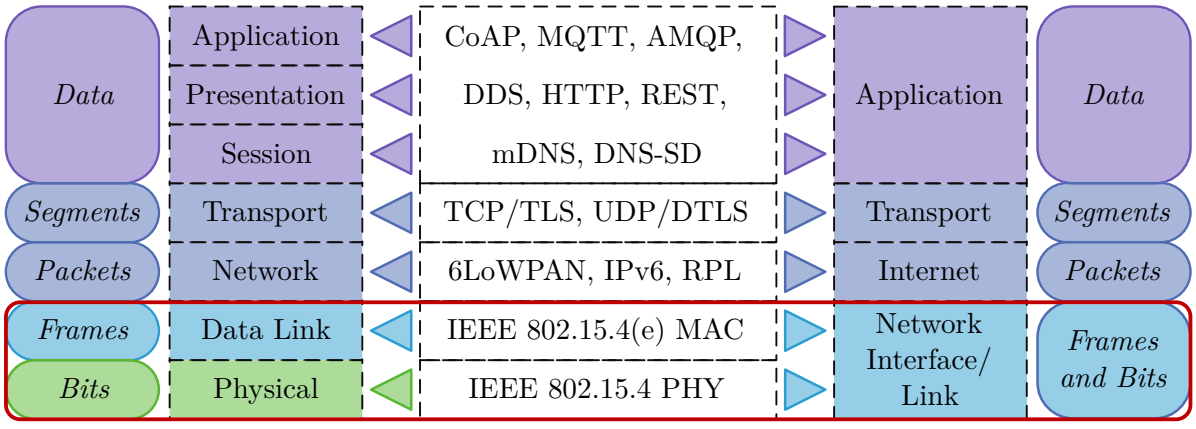
The protocol stack of individual resource-constraint wireless communication systems and the *Internet of Things* essentially deviates due to the communication protocols used. There is a large, constantly changing number of protocols which appear frequently, while others disappear into obscurity. The network link layer of the protocol stack, e.g., plays a major role in delivering satisfactory results for the performance of wireless networks. Especially in the area of industrial process automation, safety-relevant and time-critical control systems are increasingly affected. So, there are definitely many questions that arise in advance when implementing these systems. Some of the current research issues will be presented hereafter, but one of the most fundamental questions underlying the motivation

of this thesis is: *How to design and implement novel protocol schemes, innovative protocol stack approaches, or optimized transmission methods for wireless communication systems as a realistic lab prototype to gain confidence in its resilience and usefulness without the technology already being implemented and widely used?*

In the following subsections we will introduce dominant wireless network link layer protocols and our candidate of choice for exemplary practical evaluation as well as the basic idea behind protocol-based evaluation methods for wireless communication systems. Furthermore, challenges for high-accuracy evaluation based on associated research issues are addressed, seamlessly transitioning to the objectives and contributions of this thesis.

Wireless Network Link Layer Protocols

Wireless sense and control networks and the *IoT* are embedded in an ecosystem of different protocols, including international standards, as well as industry-driven, or community-oriented developments. Most of them can be classified into common architectural patterns for interoperability with other communication systems especially the Internet. Figure 1.1 illustrates the different layers and semantics of the common models for communication architectures *Open Systems Interconnection (OSI)* (a) and *TCP/IP* (c). Exemplary relevant protocols used in the protocol stack for the *Internet of Things* are listed in Figure 1.1 (b). The *link layer* for wireless technologies is particularly affected by this diversity, since it is required for every type of physical communication and tends to be technologically driven by a variety of industrial players. Because it is rather confusing to list these in this figure as well, we apply here a rough classification of some relevant technologies, protocols, and standards. In contrast to this, we highlight *IEEE 802.15.4* as the standard we chose for our experiments.



(a) OSI Reference Model (b) IoT Example Protocols (c) TCP/IP Stack

Figure 1.1: Layered protocol stack architecture and semantics of protocol data units for **(a)** *OSI* and the **(c)** *TCP/IP* stack with **(b)** *IoT* example protocols.

Because of different semantics of layering in the *TCP/IP* protocol stack and the *OSI* reference model, the *network interface* or *link layer* designates the lowest layer of the internet protocol suite and a combination of the *Data Link Layer (DLL)* (*layer 2*) and the *Physical Layer (PHY)* (*layer 1*) in the *OSI* model point of view. Network link layer protocols are typically hardware-dependent and provide an abstraction for the higher layer protocols (e.g., *TCP/IP*) that are to run independently at any kind of network interface. Particular protocol standards for wireless network technologies are commonly designed in *OSI* fashion and thus comprising the *Medium Access Control (MAC)* at the *DLL* and the *PHY* layer. They can be categorized according to specification complexity, feature sets, common availability, energy consumption, and typical communication ranges or data rates. Dominant link layer protocols, actively used in wireless *IoT*, industrial automation, or monitoring applications, can be roughly classified by their communication range. Comprehensive studies on current wireless network link layer protocols and network technologies [2–7] are recommended for further reading.

Short range link layer technologies cover very limited distances of a few meters up to 100 meters for *Line-of-Sight (LoS)* connections. The average power consumption and data transfer rates are typically very low.Wireless sense and control networks usually need *multi-hop* communication to propagate data by using multiple nodes. Example short range link layer protocols for *Wireless Personal Area Networks (WPANs)* are:

- *Bluetooth LE* (*BLE*) (Bluetooth *SIG*) *(short range)*
- *EnOcean* (EnOcean Alliance / *ISO/IEC* 14543-3-10 [8]) *(short range)*
- *IEEE 802.15.4* (*LR-WPAN* / *IEEE 802.15.4* [9]) *(short range)*
- *ISA100.11a* (*ANSI/ISA*-100.11a-2011 [10] / *IEEE 802.15.4*) *(short range)*
- *WIA-PA* (*IEC* 62601 [11] / *IEEE 802.15.4*) *(short range)*
- *WirelessHART* (*IEC* 62591 [12] / *IEEE 802.15.4*) *(short range)*
- *Z-Wave* (*ITU-T* G.9959 [13]) *(short range)*
- *DECT ULE* (*ETSI TS* 102 939 [14]) *(short/medium range)*

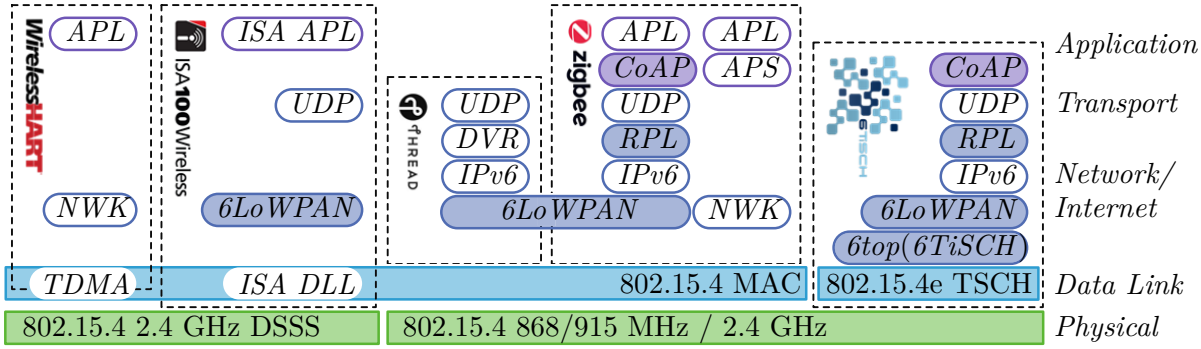
Long range link layer technologies can reach devices in urban or wide transmission areas up to kilometers, comparable to mobile phone or cellular networks. For low power and low data rates there are specialized technologies. Typically, the end devices are *single-hop* connected with a direct link to a central gateway. Example long range link layer protocols for *Low Power s (LPWANs)* are:

- *Wi-SUN* (*Wi-SUN* Alliance / *IEEE 802.15.4*g/e) *(medium/long range)*
- *LoRaWAN* (*LoRaWAN* Alliance / *ETSI TR* 103 526 [15]) *(long range)*
- *MIOTY* (*ETSI TS* 103 357 [16]) *(long range)*
- *NB-IoT* (*ETSI/3GPP* Release 13 [17]) *(long range)*
- *Sigfox* (SIGFOX S.A.) *(long range)*

In the technology chapter in the appendix we give an very brief overview and introduction to selected protocol standards for wireless network technologies, apart from proprietary solutions that still exist. The development and standardization of protocols is increasingly fast. Significant research activities at the link layer are also emerging in the field of reconfigurable *MAC* [18, 19] and *Cognitive Radio (CR)* [20–22]. One protocol standard that has in particular shaped *WSN* and *Internet of Things* developments from the very beginning, continues to play a decisive role in this context. This is the *IEEE 802.15.4*, as highlighted in Figure 1.1. It is the most relevant open short range link layer protocol standard for *WPAN* and associated wireless technologies.

IEEE 802.15.4, Associated Protocols and Technologies

The first version of *IEEE 802.15.4* was adopted in 2006 [9] and has continued to evolve and grow in importance with revisions and amendments since then. For a detailed evolution of the standard, we recommend further reading [2, 23]. As one of the most important extensions, the *IEEE 802.15.4*e [24] amendment was published in 2012 to overcome a number of limitations, enhancing and extending the functionality to the *IEEE 802.15.4*-2011 [25] protocol. The key *MAC* behavior enhancements are the *TSCH* and the *DSME*. Furthermore, there are numerous amendments for the *PHY* over the last decade [23], e.g., the *IEEE 802.15.4*g [26] in 2012.



(a) WirelessHART & ISA100 (b) Thread, ZigBee IP & ZigBee (c) 6TiSCH stack

Figure 1.2: Protocol stack architecture of *IEEE 802.15.4*, associated technologies and communication protocols

The *IEEE 802.15.4 WPAN* protocol standard is still the pre-dominant technology for *LLNs* and short range communication especially in industrial process plants but also in wireless *IoT* based building automation and consumer electronics. Since the spread across associated technologies and communication protocols is almost overwhelming, only a selection of the currently important standards based on *IEEE 802.15.4* is shown in Figure 1.2. The de facto global standards of wireless instrumentation for industrial automation *WirelessHART*, *ISA100.11a* (a), and *WIA-PA* (*IEC* 62601 [11]) extend *IEEE*

802.15.4 with additional specifications for the *MAC* layer as well as higher layer protocol functionality (cp. [27, 28]). In the *IoT* context, *ZigBee* (*IP*) [29] (ZigBee Alliance), *Thread* (Thread Group) (b), and *Wi-SUN* (Wi-SUN Alliance) are among the most successful and important technologies.

Due to the necessary interoperability and Internet connectivity intentions of wireless technologies, further adaptation and support protocols have been standardized internationally. The *Internet Engineering Task Force (IETF)* has defined protocols to enable integration of resource-constrained devices (i.e., sensor/actuator devices) into the Internet. The *over (6LoWPAN)* (*RFC* 4944 [30]) and the *6top* (*RFC* 8480 [31]) adaption protocols, as well as the *RPL* (*RFC* 6550), and the *CoAP* (*RFC* 7252 [32]) are the most important of them. In Figure 1.2 (c) the exemplary stack of *over the mode of 802.15.4e (6TiSCH)* (*RFC* 8180 [33]) is illustrated. Because most of the vendors and *Special Interest Groups (SIGs)* focus on interoperability and Internet-connectivity without incorporating custom *protocol gateways*, the *IETF* specifications are meanwhile also integrated in many stacks of proprietary technologies, e.g., *6LoWPAN over Bluetooth Low Energy (BLE)* (*RFC* 7668 [34]), or *IPv6 over DECT ULE* (*RFC* 8105 [35]).

Evaluation of Wireless Communication Systems

There are several and partially complementary design and validation tools and methods for the performance evaluation of communication technologies, protocols, and algorithms [36]. These are extremely important for the research and development of improved architectures and novel approaches for wireless communication systems and protocols [37]. The most common methods are: *analytical analysis*, *simulation*, *emulation*, *field tests* or *testbeds*, and real deployment.

Analytical analysis [1, pp. 27ff] is based on mathematical models which represent the system in terms of a set of mathematical equations. It can predict the behavior of network applications and protocols, and is used to understand phenomena in networked systems. But these methods have to make over-simplified assumptions in order to keep their models traceable. This severely limits insight into a complex system design and tends to lead to untrustworthy results. Due to the inherent functional and algorithmic demands of wireless communication systems, analytical methods are unsuitable for performance evaluation.

Network Simulation [38, 39] (ref. simulation basics in Chapter 2) techniques are representing the more traditional model-driven evaluation methods for wireless communication architectures and protocols. The network simulation approach simplifies the construction of layered communication protocol architectures and application models for networked systems. Simulation is usually the preliminary step when evaluating new protocol-based approaches for *WSNs* and the *IoT* and is often used to compare different design alternatives or to optimize given designs [38]. The major drawback of simulation is the inaccuracy

representing the *PHY* and radio channel with reasonable computational effort. To achieve an accurate representation of an entire network of wireless transmission systems, this layer must be taken into account in detail. Some simulation models of the *PHY* are presented in Subsection 2.2.2. However, these models are only an approximation of real effects and are still expensive to compute. For this reason, we have to ask questions about how to achieve a modeling of complex physical phenomena within pure simulations.

Field tests and *testbeds* [1, pp. 33ff] are used to perform automated measurements and performance tests under real environmental conditions with complete system implementations. While placing hundreds of nodes in harsh environments is not feasible or practical for field testing, testbed platforms can provide controllable environmental conditions for deploying real systems in hardware and software. This provides a basis for experiments in which the wireless network of sensor nodes is exposed to the physical influences of the radio environment. A testbed platform can serve for planning real-world deployments according to high-fidelity evaluation of sophisticated *WSN* designs [40]. Over-the-air testing of large-scale wireless communication networks, however, and correcting design errors that emerged as a result can be very time-consuming and expensive in general. Furthermore, it requires a specific setup, customized hardware, software, and environments. In practice, most of real-world-related experimentation is based on specific, custom made testbed platforms. Well-known open testbeds are used less frequently for experiments [37]. In [41] numerous testbeds for *WSNs* are introduced.

Emulation [1] (ref. emulation basics in Chapter 3) is considered as a hybrid approach in which some system components are deployed on real target hardware and some are reproduced by virtual or hardware-based models. Usually, *network emulation* aims to combine the advantages of simulation, e.g., flexibility and observability, with real-world runtime accuracy. Although there are a number of approaches for emulating networked computer systems and emulation platforms, we state that *network emulation* is far from being fully researched and understood, especially for the evaluation of wireless communication systems. Rather, this term is used for several different techniques and sometimes even misunderstood.

A detailed analysis of these techniques is required in order to precisely identify and expound problems and limitations. In addition to the following summarized research questions and challenges, we refer to a basic comparison of the aforementioned techniques in the literature [1, pp. 37ff], [38, pp. 83ff] and to the corresponding summaries in Section 2.4 as well as Section 3.4. Furthermore, we supplement the underlying motivation in this short introduction in Chapter 4 with the objectives and envisioned benefits of the approach we pursue in this thesis.

Research Questions and Challenges

The basic evaluation methods have their significant drawbacks and limitations when used in isolation. There are various surveys and sprawling discussions about *when* and *how* to apply which technique [1, 38, 40, 42–48]. A detailed analysis of these techniques is required in order to precisely identify and expound evaluation problems and limitations. For this, we refer to (Section 2.4 and Section 3.4). The most important methods are still simulations and testbed-like deployments, but only few studies apply multiple or even combined evaluation methods. In addition, pure simulation studies of wireless communication systems tend to decline, while real-world experimentation is still present [37].

The wireless link layer for *WSNs*, *Low-power and Lossy Networks* and the wireless *IoT* is still underrepresented within network simulators. Models reported in the simulation literature are designed with simplified assumptions about the *MAC* and the *PHY*, as well as the radio and channel models. Usually, the basic unit of a network simulation is assumed to be a *message* which is a *frame* or a *packet* but rarely a single *bit* and certainly not a (partial) *wave* of an electromagnetic field. Because channel models are applied to frames as a whole, considerations of the *PHY* signal processing details, such as fast fading, or frequency-selective channels as well as frame construction and reception cannot properly be accounted for. Due to this methodical abstractions at the link layer within pure network simulation, it is very difficult to study the impact of mechanisms of the physical details on higher layers. For example, *MAC* protocol prototyping requires close interaction with the *PHY* and the radio frontend [18]. Within *Cognitive Radio Sensor Networks (CRSNs)* [20] and *cross-layer design* [49] approaches, there is a necessary demand of information exchange between all of the higher communication layers with the physical communication interface. Therefore, accurate models and realistic real-time behavior for the link layer are requested in simulations [42, 50–52]. Normally, the simulation can score in terms of scalability, but the more accurate the models are, the more likely scalability will reach its limits.

Physical *testbeds* are expensive to maintain and usually rather small. They typically lack of reproducibility and flexibility [40], as they consist of a single type of sensor node, radio interface, network stack, and operating system. Present *testbed* platforms are quite different and provide a single, fixed network topology and constant or not reproducible environmental settings. The number of recent publications that uses *testbeds* or *field tests* experimentation allowing reproducible conditions is very low [37]. Compared to the large number of testbeds [41], very few popular platforms do support mobility [53], which is one of the key aspects of future wireless communication system designs. In addition, each system requires guidance to achieve scientific results, whereas a number of relevant operating parameters, such as radio interference or the transmission power, are beyond user control. Limited, small-scale testbeds will no longer suffice if the number of network devices in future wireless automation or *IoT* applications continues to grow.

More than half of the research and evaluation activities of *WSNs* are related to *MAC*, *DLL*, *PHY* and *cross-layer* simulation studies and experiments [37]. Regardless of the specific link layer protocol standard or radio technology, there is a need for concepts for massively scalable evaluation environments that can answer ongoing practical as well as research questions. As a fundamental motivation for our work, we derived (cp. [18, 20, 42, 44, 46]) and identified four key challenges involving the wireless link layer from different *perspectives* in the form of research questions for wireless communication systems evaluation:

- *For coexistence:* Because of new technologies and higher channel utilization, especially within the 2.4 GHz range, the wireless medium is becoming more and more dense. How to design and evaluate resource sharing with other collocated *WSNs* in the wireless spectrum that use different or even the same technology?
- *For network robustness and resilience:* Constantly changing environmental characteristics due to mobility and harsh radio environments are key challenges of future wireless communication system designs. How to achieve self-stabilization and what is the impact of physical node mobility, density, and fluctuation within a *WSN*?
- *For Cognitive Radio and cross-layer studies:* Decision-making processes of higher layers in *CRSNs* need detailed information about various parameters and the current state of the link layer. How to consider impacts of constantly changing physical parameters, radio effects, or *transceiver* techniques on the higher layers of the protocol stack, up to the (distributed) application?
- *For intelligent and energy-efficient media control:* Static protocol schemes are prospectively insufficient to deliver desired performance in a dynamic spectrum environment. How to achieve model-based *MAC* protocol prototyping in realistic protocol stack scenarios on accurate physical layers and realistic radio channels?

1.2 | Objectives and Contributions Overview

There are always many limitations with all these evaluation techniques, in addition to their benefits. As mentioned above, certain approaches of ongoing research issues cannot be operable evaluated by these alone. *Hybrid* evaluation techniques attempt to encompass different aspects according to concrete evaluation questions and thus make use of the advantages of several techniques. We review related work in depth and discuss the possibilities and benefits of hybrid approaches in Chapter 4.

What is desired to answer above mentioned research questions is a flexible model-driven experimental environment for a realistic evaluation with accurate *MAC*, *PHY*, and channel modeling in a hybrid manner. It should be able to overcome several limitations by physically incorporating the influence of signal propagation effects and changes in protocol flows, parameter settings, and distributed applications but without having

all components already implemented. We contribute to address related problems and research questions, and create a concept and prototype for a completely novel approach to hybrid wireless network emulation.

The main concept includes several theoretical and practical contributions from various perspectives that arise when incorporating substantially different techniques. Based on common naming conventions introduced with the state-of-the-art concepts in Chapter 4, we introduce a novel evaluation methodology, called *Split-Protocol-Stack Parallel Simulation and Emulation with RIL*. While the conceptual overview is presented in Chapter 4, the following is a brief summary of the key contributions with reference to the relevant thesis chapters and the author's publications.

- *Interface control* and *abstraction* is needed for accessing real radio transceiver hardware among all involved subsystems, e.g., from model-based simulations. We introduce a universally valid, system- and hardware-independent solution for control and data exchange based on an intermediary subsystem (Section 5.3, [54]).
- In order to enable realistic *PHY interaction* and *radio transmission* on target hardware, we propose an event-based scheduling process both on transceiver chip hardware [55] and a *Software-Defined Radio (SDR)* system [56], called *RIL* (Chapter 6).
- The *resource allocation* and *optimization* of target hardware in testbeds and systems is an other difficult task. Here we suggest a graph-based *Mixed Integer Linear Programming (MILP)* optimization approach for allocating nodes in radio emulation testbeds (Chapter 7, [55]).
- With *reference measurements* from practical *WSN* deployments emulation scenario results can be evaluated and compared. We present with practical *Received Signal Strength Indicator (RSSI)* measurements of sensor node placements in both a environmental monitoring scenario (*unpublished*) and a target emulation testbed system (Chapter 7, [54]).
- For the *synchronization* of computer simulation with real-time transmissions on the wireless medium we need to guarantee high accuracy. We recommend a new synchronization scheme, suitable to enable accurate realization of wireless transmissions (Chapter 5, [57]).
- *Scheduling* and *dispatching* events in a hybrid approach are a challenging research problem. For this we extend *Discrete Event Simulation (DES)* capabilities with a pseudo-real-time scheduling procedure [57], as a result of our *synchronization* concept (Chapter 5).
- The *implementation* of the concept as a framework-like prototype is a necessary step for experimental work and evaluation. We provide various prototype implementations of our concepts and significant system components on common platforms (Chapter 8, [54–56, 58]).

1.3 | Thesis Structure

The structural outline of this thesis is shown in Figure 1.3. We have already introduced the need for high-fidelity link layer evaluation and outlined the main objectives and contributions of this thesis in Chapter 1. The analytical part of this thesis is essentially covered by the following Chapters 2 and 3, each highlighting both the theoretical and technical fundamentals of contrasting network evaluation methods as well as the state-of-the-art in science and technology of comparable research. Furthermore, a broad overview of comparative approaches of hybrid and parallel simulation and emulation techniques for evaluating wireless communication systems is presented, which results into the conceptual overview of our approach in Chapter 4. In Chapters 5, 6, and 7 the main challenges and achievements of the approach are presented. In each chapter, conceptual and experimental aspects are outlined in detail with different theoretical and practical contributions and evaluations. Moreover, as a result of the conceptual work, Chapter 8 presents the exemplary *SEmulate* prototype system for experimentation as well as a practical demonstration of the approach capabilities based on a state-of-the-art *cross-layer* optimization case study. An executive summary of the results with an orientation for future work and research in Chapter 9 concludes this thesis.

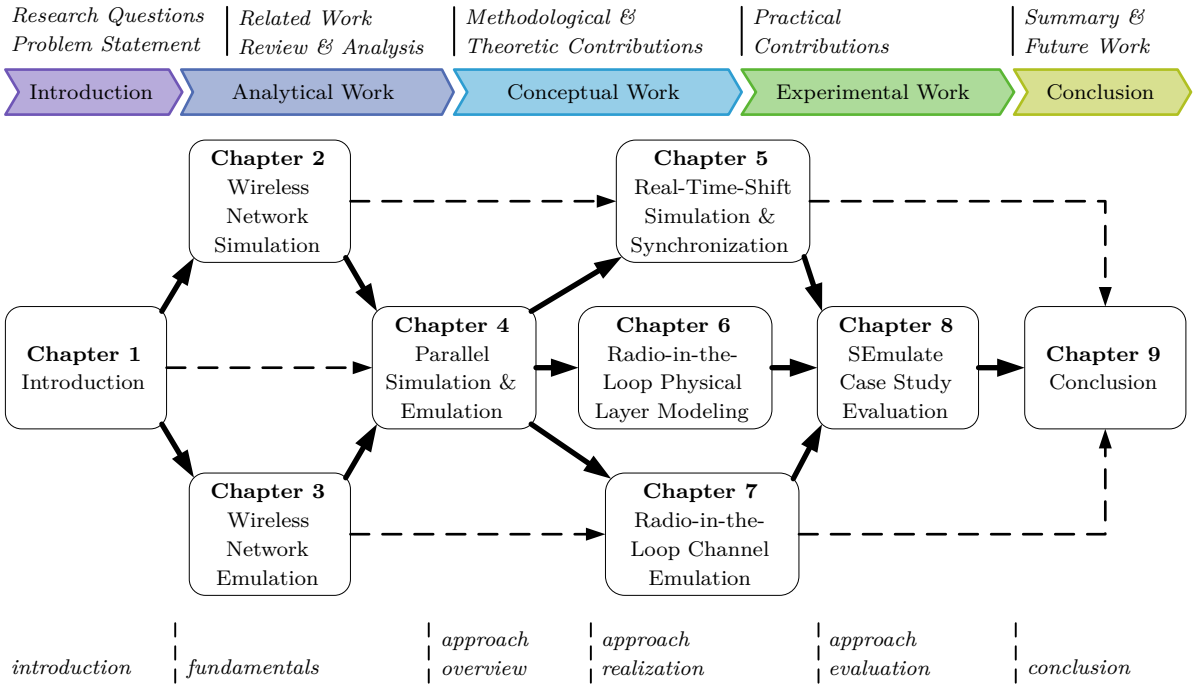


Figure 1.3: Structural outline of this thesis

2 | Wireless Network Simulation

The performance evaluation of complex and dynamic real-world wireless network systems and applications usually requires the implementation of the entire system, whereby network measurements can be performed at different vantage points, e.g., the end/intermediate systems, or the wireless links/environment. But, how to evaluate (new) algorithms, protocols, or applications without having them already implemented and installed in the supposed environment? In the network research community there is broad agreement about the vital role of network simulation in terms of *scalability*, *reproducibility*, *rapid prototyping*, and *education* (cp. [38, 46, 47, 59] for a detailed overview of this area).

In this chapter, we introduce important theoretical and technological fundamentals of (simulation-based) wireless communication network evaluation and discuss in detail modeling strategies for wireless network simulation and selected system models, which partially serve as basis for this thesis prototype implementations. Based on the analytical overview of related simulation tools and specific models, this chapter concludes with a discussion about limitations of pure network simulation systems to underline the demand for hybrid evaluation techniques in general and the *Radio-in-the-Loop* concept and prototype introduced with this thesis in particular.

2.1 | Simulation-based Network Evaluation

As briefly discussed in Section 1.1, for the most of the research questions in wireless networks, the system complexity makes pure analytical analysis impossible. The simulation, on the other hand, is based on arbitrarily accurate models which help identifying potential problems and provide estimations of performance of a system for new algorithms, parameters, policies, or different operating conditions. Thus, simulation is an efficient and cost-effective way for experimentation using different time frames, such as compressed time to speed up or expanded time to observe details of a study.

However, a simulation is only as accurate as precise the developed models are, i.e., whether they match the aspects of the real system, e.g., the protocol behavior. By retracing the message and event flow of communication protocols, simulation provides a suitable environment to evaluate the design of novel designed protocols or protocol stacks without implementing them on physical real devices to answer questions like: *How does replacing individual protocols or protocol stacks, as well as changing protocol parameters affect the performance of a network application?*

2.1.1 | Terminology and Classification of Simulation Modeling

In this thesis we consider a dynamic hardware/software system or a time-dependent information technology process as the basis for a common understanding. For the mathematical modeling of such a system in general or a wireless transmission system in particular, we also need to clarify the terminology, concepts, and delimitation. Because there is no fully standardized set of terms in the literature, we loosely adapt definitions from [1, 38, 60–62] and give an example corresponding the scope of this thesis for each. Furthermore, we do not go into detail with specific modeling techniques, they are listed and compared in [62, pp. 193ff].

Terminology

An *entity* [60, p. 30] is a particular object of interest which can be described by a variable number of *attributes* [38, p. 2]. To give an example, we can have a look at an entity *packet* in which the *header* (for addressing the communication endpoints) and the *payload* (for the data to be transmitted) can be defined as attributes.

A (complex) dynamic *system* [38, p. 2] is a set of interconnected and interdependent *entities* to create a purpose or function that is not reduced to the features of the individual entities. Seen from the outside, we consider networked computer systems as incredibly complex. Looking at different levels of abstraction though, a system can simplifying be characterized by its parts and their interactions. In the wireless communication context of this thesis, e.g., a *WSN* is a system with a set of interconnected wireless devices (sensor nodes) that communicate by sending *radio packets* to provide end-to-end connectivity. As the system is also effected by its surrounding, it is fundamental for the modeling process to determine a *boundary* between the system and its environment depending on the purpose of the problem under investigation [60, p. 30]. For each system the current observation time must be taken into account. The *state* of a system can be *continuous* or *static* (as introduced with the classification of systems and models hereafter), but it is defined by a set of variables at a certain operating time.

In general, a *model* of a system (cp. [60, p. 33], [38, pp. 6ff.], [39, p. 3]) is a representation of a specific view on that system for a studying purpose. This abstraction as a model contains only some selected features and characteristics. It is, however, necessary to consider all aspects of the system that effect the problem under investigation as complete as possible. Often it is not easy to identify which aspects and features must be taken into account [61, Sec. 5.2 or pp. 249ff]. Within the wireless communication domain, it could be very extensive to model the system comprehensive and precisely. Therefore, we have to make abstractions, simplifications, and assumptions in the modeling process. On the other hand, we also have to keep in mind that there are a lot of pitfalls when creating a model of a system. These have been summarized and discussed in [62, pp. 190f] and [38, pp. 8ff].

Furthermore, the modeling process depends on the considered system aspects and can be of very different abstraction levels. Looking at wireless communications, for example, the model of a packet transmission involves continuous signals in a seemingly infinitely complex transmission channel that includes not only the communication devices involved. In contrast, a model of a higher layer communication protocol could be a rather simple implementation of procedures from its specification. In this thesis context we focus on holistic modeling the message exchange based on the wireless devices *protocol stack*, in contras to application-specific peripheral processes, e.g., sensor readings or control tasks at the concrete vendor device hardware.

Classification of Systems and Models

Models are generally considered as mathematical representation of dynamic *systems* for the computer-aided analytical analysis. *Physical* models (cp. [61, pp. 4f] and Figure 2.1) are rather created to illustrate and understand real systems with a more natural context to the physical reality. A *static* model represents a system at a particular point in time, a snapshot, which is not in focus when modeling *dynamic* communication networks. However, a very important differentiation in the context of dynamic systems is the classification into *discrete* and *continuous* systems and models [60, p. 32] [61, pp. 3f].

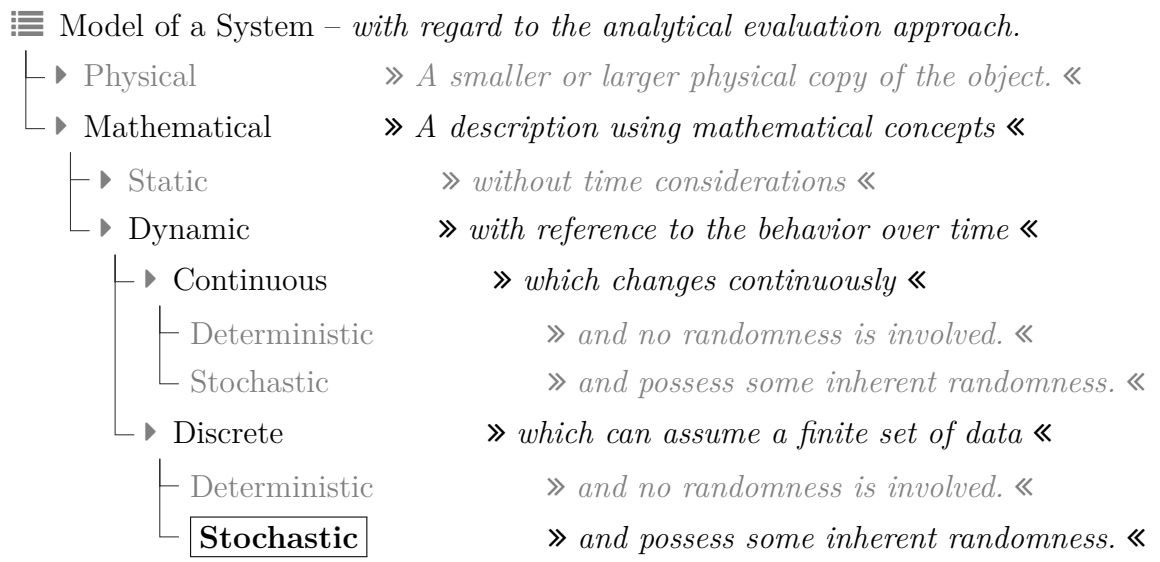


Figure 2.1: A classification of models. The relevant model characteristics for this work are highlighted.

If we look at a *discrete* system, state variables change instantaneously at different points in time of system execution. What happens between the state changes is not relevant for the further execution of the system. A *continuous* system changes its state variables in infinitesimal small time steps during the execution. Most systems in which a single type

predominates can be classified as either discrete or continuous; others can be considered as *hybrid* systems.

Analogous to the classification of dynamic systems, models (of systems) can also defined as discrete or continuous. Discrete simulation models, however, are not always used to model a discrete system (the same also applies to continuous models). Which type of model is selected crucially depends on the system characteristics and the objective of the studies. For example, a continuously changing communication channel can be modeled discretely if only the change of the properties (e.g., the packet error rate) and the time-scheduled sending and receiving of individual messages are of relevance. When modeling complex systems, it is also conceivable to use continuous and discrete models in combination [61, p. 713]. Moreover, communication processes should be modeled *stochastically* to introduce some randomness because many properties are based on probability distributions.

The overview of models in Figure 2.1 recaps the classification characteristics explained above. Simulation models in the scope of this thesis are referred to as dynamic, discrete, or stochastic (as highlighted), in the following: » *A description using mathematical concepts with reference to the behavior over time which can assume a finite set of data and possess some inherent randomness.* «

2.1.2 | Network Simulation Methodology

Network Simulation is used to perform evaluations and measurements on models of communication systems. Simplified, a network simulator is a software process (on specialized or general purpose hardware) that generates outputs from given inputs. As depicted in

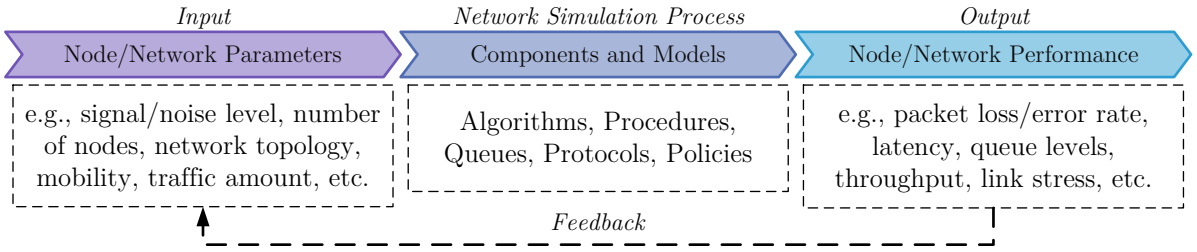


Figure 2.2: Network simulator abstraction

Figure 2.2, the network simulation process runs a set of algorithms, protocols, policies, and others, representing the evaluation models. The input sequence of a wireless network simulation can be quite diverse according to the research question, but some of the typical model parameters are listed in Figure 2.2, e.g., physical channel properties like the signal and noise level, the network topology, traffic and mobility patterns, or network dynamics that include node and link failures (ref. Section 2.2 for an in-depth overview).

The result of the simulation is a representation of the behavior of the network based on performance parameters which can have a dual purpose ([62, p. 192]). On the one hand, this serves to validate and verify the implemented network models, especially since, unlike many black box simulation systems, a network simulator can often visualize and record the actual process flow of the simulation, e.g., the exchange of messages between model entities, processes, and communication devices, to analyze and understand it in detail. This enables, on the other hand, analyses of the specific network performance behavior, regarding, for instance, the packet loss rate, end-to-end latencies between time-aware network applications, or the utilization and throughput on individual links.

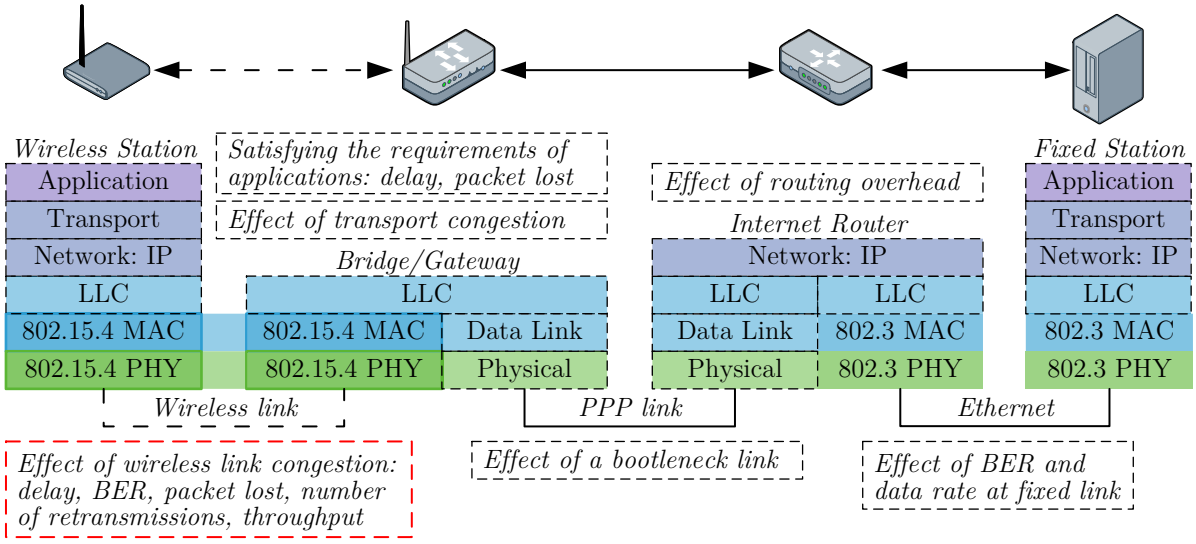


Figure 2.3: Requirements and effects in communication networks evaluation - a protocol stacks perspective with relevant example protocols (inspired by [63])

The outcomes of the communication network evaluation from the protocol stack's point of view is shown in Figure 2.3, which also includes example protocols. When looking on the different protocol stack layers, there are different requirements and effects in network evaluation. Since this thesis considers lower layer wireless devices network simulation, the special focus is on the effects of wireless link congestion, e.g., packet delay, *Bit-Error Rate (BER)*, packet lost, throughput, or the number of retransmissions.

Similar to simulation models, dynamic simulation systems can be classified in continuous and discrete ones. *Continuous Time Simulation (CTS)* applies to system models that continuously traces the system behavior according to a set of (typically differential) equations. *DES* is the most common technique when simulating communication networks based on communication protocols. This is because communication protocol specifications allow to avoid system states between two discrete times. In the following we focus on the latter here.

2.1.3 | Discrete Event Simulation

The dominant simulation technique in context of networked computer systems is *DES*. This subsection represents the most important fundamentals referred to as basis in the later chapters. For an in-depth introduction of *DES*, we recommend [38], [60], [39], and [62].

The Basic Principle

The basic principle behind the *DES* [38, p. 3][1, pp. 30-31] paradigm is to handle system *events* during the simulation execution. In communication systems a simple event (often referred to as *event message* or just *message*) is represented, for example, by triggering a packet transmission or the expiration of a timer. The execution sequence can also compose complex events, e.g., receiving an application layer request over the protocol stack. Nevertheless, each event may trigger specific system state changes and thus the generation of new events to be executed in future. Simplified, the simulator jumps from one event to another by increasing the *simulated time*, explained in the following. This is in complete contrast to *CTS*, where the simulation applies to represent a continuously changing system. Usually, *DES* completely decouples the simulated time from the continuous real time. In Figure 2.4 you can see the state changes with events (e) at discrete points over the time. With every event, the simulator creates a snapshot of the system state, containing required data to progress the simulation (cf. [38, p. 3]). Furthermore, the abstract simulation time has to be stored in a system variable as real number. It is increased with the time of the next event. Although the event execution is sequential, even for events with a high amount of CPU time, the simulation time is equal for all events scheduled at the same time.

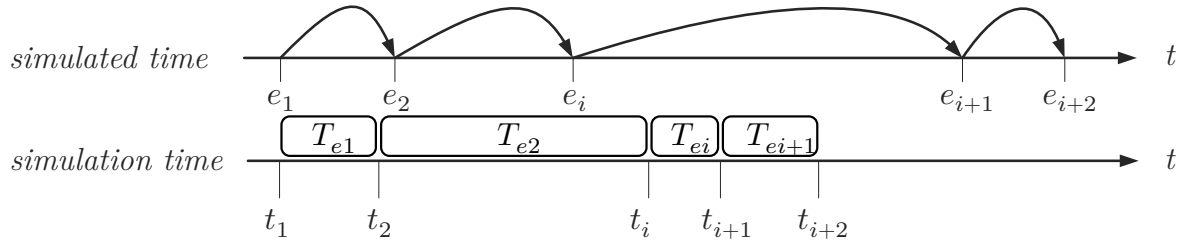


Figure 2.4: System state changes are triggered by handling event messages that occur at discrete points in *simulated time* (based on [38, p. 3]).

All events of a simulation run need to be managed in an appropriate data structure called *Future Event Set (FES)* or list (*FEL*). Simulator implementations use schedulers to retrieve the next pending event from the *FES* (sorted by the event time), advance the simulated time, and delegate the event processing to the corresponding *handlers*. This is exactly why each event must be composed of at least two entries (*time*, *type*).

While the *time* indicates the event occurring, the *type* refers to the handling procedure. Additional information is given with parameters for the handler routines. Simple *DES* engines only need to be single-threaded because each event time can be calculated which makes simulations reproducible and repeatable. The Algorithm 1 shows exemplarily the core of a *DES* engine: the event-scheduling and -processing (cf. [60, p. 148], [38, p. 4]). *DES* extensions are mostly accompanied by adapted scheduling strategies, e.g., real-time.

Algorithm 1: Event-Scheduling Algorithm

```
Precondition : Initialize simulation model, simulation time 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
```

Real-Time Discrete Event Simulation

In contrast to non-real-time *DES*, where the simulation correctness only depends on the deterministic event processing according to the *FES*, in real-time simulations the advance of the simulated time is bound to the continuous wall-clock time. As there are different naming conventions for the different concepts of time, the following definitions are made according to Fujimoto et al. [64] and Wehrle et al. [38]:

- *Simulation* or *physical time* is the time flow introduced by the execution of the event procedures on the simulator host's *CPU*.
- *Simulated* or *virtual time* is the abstract value according to event timestamps.
- *Wall-clock time* is the elapsing real atomic time, regardless of the simulation or the simulated time.

The term *real-time simulation* in the field of *DES* appears in the literature in connection with emulation features of simulation environments (cp. [65], [66], or [67]). In general, it can be stated that real-time simulations are always used when the simulation system is coupled or has to interact with other non-simulated (at least real-time dependent) systems. The execution of a simulation model (of a physical system) then occurs at the same time rate as the actual physical system.

In real-time simulations the *simulation time* is equal to the *simulated time*. The time a simulator takes to complete all changes of state variables for an event execution varies depending on the complexity, effort, and the amount of computations to be performed. Therefore, two different situations can occur, as shown in the Figure 2.5. We can simply assume that a simulation is real-time capable, if the event execution of every single event is faster than the real time steps. If the execution time overruns the designated real time step, the simulation is not real-time.

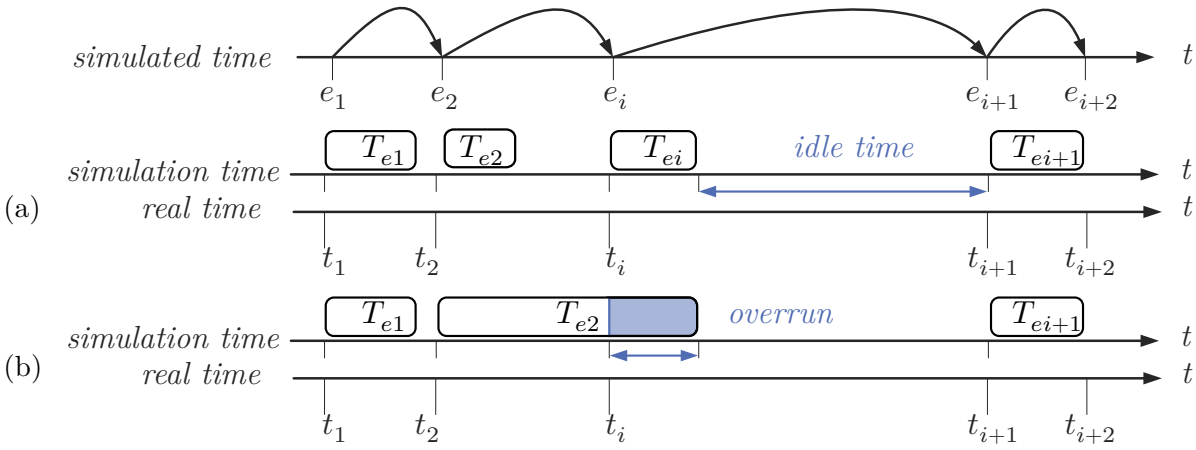


Figure 2.5: Behavior of a real-time simulation. **(a)** The event execution time is smaller or equal to the real time step. **(b)** The simulation needs more time to compute the event execution than the designated time step. The simulator can not schedule e_i because this event is already in the past.

If simulation models are configured to run in real-time, it is possible to include real hardware systems in the process. A real-time scheduler works differently than those in *DES*. While the *Event-Scheduler* takes all events out of the event queue one after the other and processes them immediately, the *Real-Time Scheduler* does not perform processing until the time of the event is reached. The use of a real-time scheduler is an elementary basis for a simulation coupled with real hardware.

2.2 | Modeling for Wireless Network Simulation

Modeling refers to the creation and configuration of simulations, e.g., defining the network topology, interconnecting models to represent different system functions, or defining model parameters for systems and their network links. On the other hand, it comprises the development of concrete models for specific network protocols and functions. The latter is the focus of this section, which serves to present a basic state-of-the-art overview about discrete-event modeling for wireless devices network (protocol) simulation as well as to delimit the scope of this work. The following subsections present important insights and key modeling practices on the substantial different layers and domains (cp. [38, p. 137]). For a further decomposition of wireless communication systems and networks we assume several domains and specifications besides the layers, based on common link layer protocol specifications (ref. Section 1.1) and wireless transceiver designs. In Figure 2.6, an overview about modeling layers and domains as well as used terms and conventions is given.

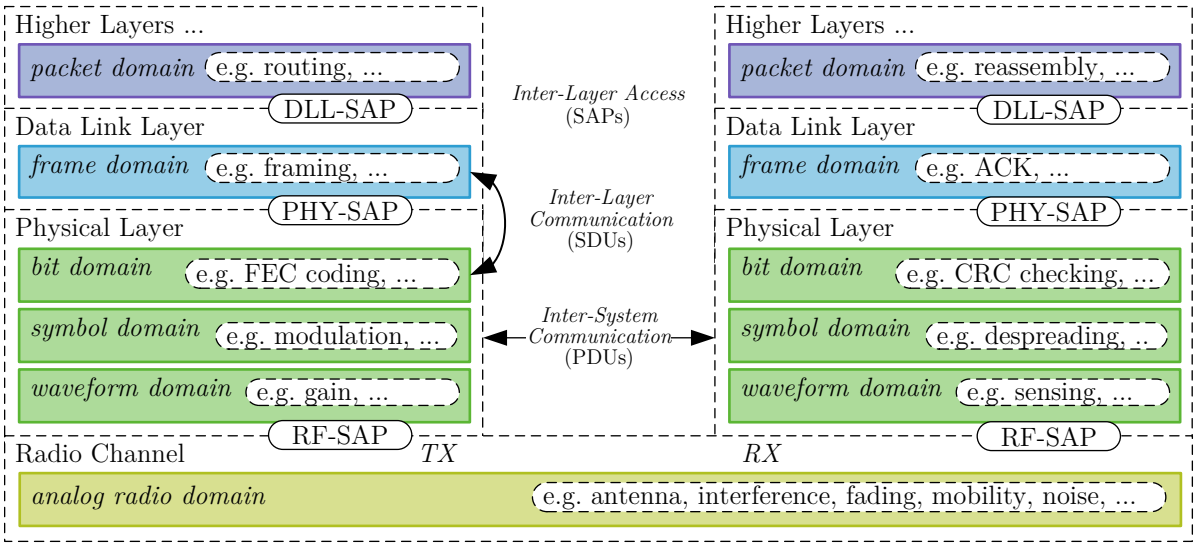


Figure 2.6: Decomposition into protocol stack layers and modeling domains of wireless communication systems for sender and receiver

Models of communication protocols and their layers (ref. Section 2.3) do not generally apply to this formal made conventions and do not necessarily define all interfaces (*Service Access Points (SAPs)*) and data units in the protocol stack. Especially in link layer protocols, inter-layer communication between *MAC* and *PHY* (based on *Service Data Units (SDUs)*) is often realized by function calls without modeling *SAPs*. Due to inter-system communication interoperability, modeling the general *Protocol Data Unit (PDU)* frame structure and (selected) management units has to be taken into account at least.

2.2.1 | Radio Channel Modeling

The *analog radio domain* (so-called propagation channel) of a wireless communication system is affected by massive impairments. A good understanding of the radio channel with its widespread effects is a key capability, when enabling the performance of a system evaluable. This area is typically constrained, highly abstracted, or sometimes omitted completely in network simulations, but there are research activities in simulation models for radio channel effects to cover this domain within *DES*.

Propagation models can roughly be classified into *deterministic* and *empirical* approaches (cp. [38, Sec.11.2, p. 199]). Deterministic models are based on field and wave theory, as well as geometrical optics (e.g., ray-tracing) algorithms. These approaches are very expensive in terms of computation complexity, time, and memory consumption because the simulation needs to discretize the radio environment into a grid and model all objects with its surfaces. This is only applicable for very small scale simulation scenarios, e.g., calculating the indoor coverage of a transmitter, or computing radiation patterns of an antenna. On the other hand, empirical path loss approaches [38, Sec. 11.5, pp. 203f] are based on statistical characterization from extensive, specific measurements and cannot be generalized therefore. The *channel quality* within network simulation models often assumes a fixed state (referred to as *static channel*), whereas a dynamic channel model takes several parameters and properties of the *analog radio domain* into account.

- **Large-scale fading** or **path loss** modeling is the most encountered property. The radio channel is modeled according to the attenuation of the analog signal along its propagation distance from a single sender to the corresponding receiver(s) at *LoS*, denoted by h^2_{pl}. For example, a simplified uniform spread of energy in free-space is given by the path loss model in Equation 2.1, adapted from [68, Sec. 2.5].

$$FSPL_{\text{dB}} = -20 \log_{10} \left(\frac{4\pi d}{\lambda} \right) \tag{2.1}$$

λ – transmission channel center frequency wavelength
d – the distance between sender and receiver

- **Shadowing** models the deviations of the signal path loss according to the propagation environment, denoted by h^2_{sh}. Thus, the radio environment with all kinds of obstacles and materials is of crucial importance.
- **Small-scale** or **fast fading**, denoted by h^2_{fad}, of the transmission gain results from multipath propagation, which means that the signal at the receiver is the sum of several signal components that followed different propagation paths. When combining these three basic effects, a commonly used model can be derived with the following formula for the channel gain:

$$h^2 = h^2_{\text{pl}} + h^2_{\text{sh}} + h^2_{\text{fad}} \tag{2.2}$$

- **Propagation phenomena** in further forms affecting the radio wave in multiple forms. The basic effects are *reflection and refraction*, *scattering*, and *diffraction*. This effects are really hard to model. They usually are based on ray-tracing technologies, which assume massive computing power and simulation time to achieve a certain degree of accuracy.
- **Mobility** is a further very important factor which influences all other radio channel parameters according to varying positions within the radio environment. Modeling physical node mobility is an individual and complex topic of interest. It is usually done using special mobility models and movement patterns based on positions usually abstracted in two-dimensional space (cp. [38, Ch. 14] for scenarios, categorization, and examples of mobility models). There are significant effects on almost all higher layers (ref. Subsection 2.2.5).
- **Noise** is always present in the radio environment, while it has several additive sources, is of a stochastic nature and varies with time. There can be either naturally generated thermal noise from atmospheric disturbances and electronic circuity, or noise from human-made machinery. The noise power P_{dBm} in decibels relative to $1mW$ depends on the ambient temperature T and the bandwidth Δf of the channel and can be calculated as follows, whereby k_B is the Bolzmann [68] constant:

$$P_{\mathrm{dBm}} = 10 \log_{10}(k_B T \Delta f / 1\,\mathrm{mW})\ \mathrm{dBm} \tag{2.3}$$

- **Interference** is caused by radio signals from other emitting sources, e.g., due to the reuse of certain spectra of the available bandwidth from different network operators and link layer technologies (especially in the unlicensed 2.4 GHz *Industrial, Scientific and Medical (ISM)* band) and mainly splits into two parts. On the one hand, *co-channel interference* is described as a crosstalk from radio transmitters using the same the radio channel, which is an important factor limiting the overall wireless network performance. Otherwise, *adjacent channel interference* appears when transmissions happen on closely neighboring frequency bands (channels).
- **Antenna** impact is an other important factor on the radio channel. A very easy model assumes a so-called isotrophic antenna, which uniformly radiates the waves in each direction, and thus adds an additional channel gain from both the transmitter A and the receiver B to the *Receiving (RX)* power:

$$P_{\mathrm{dBm}} = P_{\mathrm{tx,A}} + h^2(t) + G_{\mathrm{Ant,A}} + G_{\mathrm{Ant,B}} \tag{2.4}$$

These modeling parameters result in a radio channel behavior on the radio packet domain in terms of *packet throughput*, *packet delay*, and *bit-error rate*. The channel quality can be measured in a simple form by the *Signal-to-Noise-Ratio (SNR)* which enables bit-error calculation depending on the modulation scheme and the transmit power P_{tx}. The

bit-error rate can then be used to calculate the packet error rate and overall throughput. As the *SNR* [38, pp. 158ff] is defined as the ratio between the received power and the sum of the impairing power sources, it can be calculated with the equation in Equation 2.5, whereby h^2 is the channel gain and σ^2 is the background noise power.

$$\gamma = \frac{P_{tx} \cdot h^2(t)}{\sigma^2} \tag{2.5}$$

Approximate formulas or curves taken from real world experience (e.g. in [69]) can provide a *SNR*-to-*BER* mapping (e.g. with a simple lookup table in [38, p. 159]). Once the packet error probability is determined by the *BER*, the simulation can perform a random decision according to the packet error rate threshold and mark the packet to be erroneous (cp. [38, p. 159]). If the *PHY* adds *Forward Error Correction (FEC)* to the radio packet, single or even multiple bit-errors can possibly be corrected. Thus, this simple random decision can be modified either according to an increased *SNR* or by obtaining a detailed lookup table considering the *FEC* coding in combination with the modulation scheme.

2.2.2 | Physical Layer Modeling

In the context of wireless communication systems, the literature often does not clearly distinguish between the *PHY* in terms of a node's protocol stack and the physical (wireless) transmission as a whole (e.g., in [48, Sec. 3.3], *Physical Layer* models include antenna, propagation and the radio medium). Accordingly, in the following the *PHY* can be considered as protocol *layer 1* with respect to the *OSI* reference model architecture (ref. Figure 1.1 (a)), a physical interface to convert streams of bits into radio waves and vice versa. Modeling the *PHY* [38, Ch. 9] [39, Ch. 3] must consider various parameters and features in different *domains* (ref. *bit*, *symbol*, and *waveform* in Figure 2.6) to ensure identical bit data when transmitting from the sender to the receiver over highly error-prone analog radio channels. Our division into this three domains can be further enhanced in modern transmission systems (see [38, Fig. 9.1]), but serves to facilitate understanding and is quite sufficient to illustrate the modeling effort of this layer.

All data frames from the *DLL* enter the *bit domain* [38, pp. 136ff], in which the functions *Cyclic Redundancy Check (CRC)*, *FEC*, and interleaving are performed. E.g. modeling the position of a bit error can be very important for higher layers (header vs. payload, according to the application). As these operations can be studied in the above literature, all of them are realized on the binary data information which is no sufficient to model a wireless transmission over the analog radio channel.

As in the analog radio domain information need to be transmitted over analog signals, the *symbol domain* serves as an intermediate step, transforming one or several bits into a data symbol, which can than represented with characteristics or parameters (amplitude,

phase, and frequency) of a (sine) wave. Furthermore, most *PHY* specifications of wireless transmission systems add additional particular symbols to a data packet as preambles for packet detection. To achieve better performance and reliability, there are further functionality added in modern modulation schemes, e.g., spreading, space-time coding, multi-carrier transmission, or pulse shaping. It is to be mentioned, that this processes are usually performed by specialized hardware on *System-on-Chips (SoCs)*.

The *waveform domain* assumes the tasks to covert and amplify between the digital information and the analog antenna interface. Typical features, realized with additional hardware components on the radio chips, are: sensing, packet synchronization, *Digital-to-Analog (DA)/Analog-to-Digital (AD)* conversion, or interface up/down conversion.

The most important *PHY* features, responsibilities and requirements of the radio transceiver for packet *transmission* and *reception* are summarized and presented in an overview below, according to the modeling domains in Figure 2.6 and present link layer specifications (e.g., *IEEE 802.15.4* [9], *ITU-T* G.9959 [13], *ISO/IEC* 14543-3-10 [8], or *ETSI TR* 103 526 [15]) in Section 1.1.

- **Packet Format and Coding**: Depending on the *PHY* specification of a protocol standard a *Physical Layer Protocol Data Unit (PPDU)* format specifies packet header, payload, and trailer fields for *synchronization*, *decoding* and *error control*, e.g., a *preamble* or a *frame delimiter*. The packet format may change within the same protocol standard in different *PHY* definitions, mostly depending on the modulation method (cp. *IEEE 802.15.4*, or *ITU-T* G.9959). Furthermore, *PHY* usually specifies bit operations (mostly used for header protection) to detect (*CRC* codes) or correct transmission errors (*FEC* codes).

- **Modulation** and **Demodulation** denotes the process of transforming bits into (parameters of) a sine wave and back. The key parameters are the amplitude, phase, and frequency. Popular modulation techniques in wireless transmission standards have a major impact on the quality and performance of the transmission, e.g., *Amplitude Shift Keying (ASK)*, *Binary (BPSK)* or *Quadrature Phase-Shift Keying (QPSK)*. As mentioned before in Subsection 2.2.1, detailed modeling the modulation is usually simplified and the most models translates a determined modulation-dependent channel quality into *BER* based on lookup tables. Thus, there is no modulated radio wave simulated within the network simulation.

- **Carrier Sensing**: feature detection or energy measurement reported as *Channel State Information (CSI)* of the *Clear Channel Assessment (CCA)* service to the *DLL* for particular *MAC* protocols, e.g. *Carrier Sense Multiple Access Collision Avoidance (CSMA-CA)* in *IEEE 802.15.4*. Based on the carrier sensing information, the *MAC* decide weather to access the channel or not. Also the a carrier sensing process adds a delay, which has to be taken into account instead of being omitted in network simulation [38, p. 171].

- **Packet Detection** for receiving a message. Depending on the transceiver state, e.g. the radio can only receive a radio packet if the transceiver is in receive state and is not already receiving another packet. Furthermore the reception of a frame depends on the detection of a signal *Energy Detection (ED)* within the current channel (e.g. signal energy reaches a certain threshold and is above the noise level). Based on the *SNR*, a *Link Quality Indicator (LQI)* for received packets can be calculated The receiving process can be abstracted and modeled with certain decisions on an "'incoming"' radio packet (cp. receiving process in [38, pp. 168ff, Fig. 9.10]), based on appended meta data.

- **Packet Synchronization** is needed onto receiving a packet of the current transmission in exact timing which is achieved with *preambles* (cp. the detailed presentation about the importance of the preamble particular for frequency-hopping radios and *Multiple Input Multiple Output (MIMO)* systems in [39, pp. 59, Sec. 3.2]).

This features additionally results in different *transceiver states* and *parameters*, such as *transmission power* and *receiver sensitivity*, channel and *frequency selection*, or the *activation* and *deactivation* and *state* changing, (e.g., *send*, *receive*, and *sleep*) of the radio transceiver. Switching the receiver state from one to another requires time, which depends on the radio hardware *SoC*. Although transceiver parameters can vary widely, there are some established values that are quite similar in the field of short range link layer technologies. Timing- and frequency relevant parameters like the maximum *Transmitting (TX)*-to-*RX* turnaround time, *TX* bit duration error, or frequency tolerance are of course dependent on channel frequency, local frequency operating regulations, as well as the modulation method and can therefore not be generalized. Some power parameters and common value ranges from different protocol and transceiver chip specifications are presented below in Table 2.1.

	Minimum	**Maximum**	**IEEE 802.15.4 O-QPSK**	
RX sensitivity	-100 dBm	-50 dBm	-85 dBm	
TX power	-20 dBm	20 dBm	-3 dBm	
CCA threshold	10 dB	20 dB	10 dB	*(above sensitivity)*
RX antenna gain	1 dB	18 dB	6 dB	

Table 2.1: Common *PHY* transmitter and receiver parameters and value ranges, compared to typical *IEEE 802.15.4* O-*QPSK* chip hardware configuration.

Modeling from discrete data bits to continuous modulated waveform signals is hard to achieve within a single evaluation environment. The limitations of pure virtual simulations are discussed at the end of this chapter in Section 2.4.

2.2.3 | Data Link Layer Modeling

In network devices, the *DLL* (ref. Figure 2.6) is often implemented in software, either as device drivers or as firmware for dedicated network interface hardware (*Network Interface Controller (NIC)*). Modeling the *DLL* (cp. [38, pp. 173ff] and [39, pp. 78ff]) for network simulation is often included in connection with the *NIC* network access, which means that it is not explicitly defined as a layer with independent communication protocols, but more often as a communication technology or network interface/link (according to the TCP/IP protocol stack in Figure 1.1 (c), e.g., the *NetDevice* abstraction in ns-3, or the link layer in *OMNeT++*/INET). This may also be one of the reasons why in the literature the modeling of the *PHY* often actually means the radio channel (ref. Subsection 2.2.2), but there are of course many models that make a precise distinction on the link layer. However, regardless of the assumed layer division, the modeling refers to the domain of a *frame*, i.e., mechanisms and functions are generally represented by the data *PDUs* (frames) of the link layer specification and not by single data bits.

As there are several different tasks and functions assigned to this layer, it is often classified into *MAC* and *link control* functions. According to its architectural embedding, a link control sublayer provides a uniform interface between the network interface hardware below and the upper pure software layers above. Such a link control abstraction layer can be found in almost all open standardized and partly proprietary protocols for wireless link layer technologies analyzed in this thesis (ref. Section 1.1, e.g., *Logical Link Control (LLC)* in *IEEE 802.15.4* [9], *LLC* in *ITU-T* G.9959 [13], *Logical Link Control and Adaptation Protocol (L2CAP)* in *BLE*, or *Data Link Control (DLC)* in *ETSI TS* 102 939 [14]). The features of this sublayer can be summarized as follows, but are not always assigned the same. At least an interface *SAP* is defined here which describes the services that the link control can request from the specific *MAC* sublayer below it.

- (De-)**Multiplexing** of incoming data, e.g., a default application data interface for non *IP PDUs*, or a *6LoWPAN* interface for *Internet Protocol version 6 (IPv6)* frames, is a very common feature to allow interoperability with various higher layer protocols within the node's protocol stack. Multiplexing is, for instance, one of the modeling features of the current *OMNeT++*/INET framework [48, Ch. 2] realization in which *dispatcher* modules are inserted between layers allowing for many-to-many or many-to-one communications between protocols (cp. [48, Fig. 2.2, p. 60]).
- **Error control** is often assigned to the link control sublayer, or can also be found also a *MAC* feature in common protocol specifications. The *Packet Error Rate (PER)* is one of the most important performance parameters of the *DLL*. As *Automatic Repeat Request (ARQ)* is usually assigned to the *transport layer* in which packet loss is caused by overload on intermediate systems, in wireless networks these mechanisms can be also found at *DLL* transmissions, since the wireless channel is very error prone (e.g., *ARQ* in *IEEE* 802.11). Often used simplified modeling

approaches are based on stochastic retransmission probabilities (cp. [38, Sec. 10.2.2, pp. 186ff], e.g., either to add additional transmission delays or calculating the statistical overhead introduced on the *PER* by retransmissions).

- **Segmentation** and **Reassembly** of exchanged *PDUs* is another important feature because the amount of data that might be transmitted within a single frame transmission varies with various *PHYs*. Thus, splitting data units into pieces (fragments) of acceptable size is not negligible and modeling has to take care about the impact on the transmission delays and resource utilization.
- **Queuing** is used to delay transmission when more data is provided than can actually be transmitted over the capacity of the wireless link. Queuing is one of the main application areas for *DES* (cp. [60, Sec. 2.3]) and usually considered when modeling for network performance evaluation to obtain statistics of the queue size.
- **Management** and provision of applications and performance specific services are less frequently found in the specifications and models, but they are typically assigned to this sublayer (e.g., unicast/multicast, and, *Quality of Service (QoS)*, or, neighbourhood discovery, and handover).

Accessing a wireless broadcast medium is one of the most challenging tasks in wireless networks, which makes the *MAC* a broad research area with its very own modeling requirements, domains, and simulation platforms (cp. Subsection 2.3.2, e.g., [48, pp. 451ff] and [70]. This means, a single packet transmission from a single node must result eventually in a (multiple, when including multipath propagation) packet receipt at any node within the transmission range of the transmitter. Considering events in a *DES* setup, this is often achieved by creating multiple *PDU* message events for all nodes in actually only a single transmission. While often only the data *PDUs* with their address information and payload are taken into account, comprehensive modeling of the *MAC* functions is a very elaborate task which requires a detailed well-considered strategy.

The *MAC* sublayer handles all accesses to the physical radio channel and is responsible for frame validation, acknowledged frame delivery, and retransmission. In the following we only present an excerpt of the most important and popular functions/tasks:

- **Packet Format and Coding**: Comparable to the *PHY*, there are also conventions for the *PDU* at the *MAC* sublayer, the *frame* format. *Medium Access Control (MPDU)* definitions are always present in link layer specifications to enable end-device *addressing*, payload data, and operation control *coding*. Usually, the *MAC* frame format classifies different frame types for signaling/management and payload data exchange. In network simulation modeling, the *MAC* sublayer is often abstracted to this (data) frame format (not binary coding, but meta information).
- **Radio Resource Management** is a common task of the *MAC* sublayer (cp. [38, Sec. 10.1.5, pp. 181ff]) because the *PHY* cannot know whether a received packet is intended for the given node. The general purpose is to save energy when transmitting and receiving, e.g., by setting the transceiver gain and amplification, or put the

radio into sleep mode.

- **Multiple access** is the most challenging and extensive research area in wireless link layer technologies with a long history. They also occupy a significant portion of the entire WSN field (cp. [37]). Multiple access techniques constitute the main part of a *MAC* protocol. They are substantially developed to avoid *collisions* when transmitting and can be roughly divided into three categories (cp. [71, pp. 15ff]): *fixed assignments* (*FDMA*, or *TDMA*, e.g., channel hopping, or handling and maintaining *Guaranteed Time Slot (GTS)* mechanisms with network *Beacons*), *random access* (e.g., ALOHA, or *CSMA-CA*), and *demand assignment* (e.g., *polling*, or *token passing*). Modeling ...
- **Management** functions are provided according to operation modes and device types of a certain *MAC* specification, e.g., *Personal Area Network (PAN)* association and disassociation, notification, and device security. These are difficult to generalize within network simulation modeling.

These features eventually result in partially huge *DLL* models when incorporating all specified details. The simulation of *MAC* protocols can be considered as an independent domain in the field of network simulation when looking at current simulation models (ref. Subsection 2.3.2). Especially for the accurate performance evaluation of *multiple access* techniques, the core of any *MAC* protocol, a close and accurate interaction with the *PHY* is required, which provides the necessary channel information and performs the corresponding physical processes. Common pitfalls are discussed in [39, pp. 92ff].

2.2.4 | Higher Layer Modeling

The modeling of the higher layers of a node's protocol stack is generally a well-established domain in network simulation (cp. [38, pp. 357ff] and [39, pp. 97ff]) and is only marginally considered in the context of this work, but of course it also plays a crucial role in the performance evaluation of a wireless network. As communication protocols are usually implemented in software running on the nodes of the host system, the requirements are more of an algorithmic nature that can excellently be served by *DES*. The processing is based on data packets at the *packet domain* (ref. Figure 2.6 accordingly). Using real implementations with special *wrappers* that interact with simulators is an often chosen approach, especially for the higher layers With the *Network Simulation Cradle (NSC)*, real-world node software is included, to be executed in simulation software. The central features and functions of the higher layer modeling are usually assigned to the established protocol layers, namely *network*, *transport*, and *application* layer and their communication protocols (ref. Figure 1.1, e.g., *6LoWPAN*, *Routing Protocol for Low-power and Lossy Networks (RPL)*, *TCP*, and *Constrained Application Protocol (CoAP)*):

- **Adaptation** introduces features to support protocols from the internet protocol suite to work with short range link layer technologies (e.g., *header compression*

or *fragmentation*), very often realized as a sublayer in the *Network Control Layer (NWK)*, but can also reside in the *DLL* as link control feature. The most important adaptation protocol is *6LoWPAN* (cp. [72] for a *OMNeT++*/INET simulation model based on a real implementation, applying the *NSC* approach), which is indispensable especially in the *IoT* world (e.g., *6LoWPAN* over *BLE*, or the *IEEE 802.15.4* based stacks Figure 1.2). Through *IETF* standardization efforts, a separate *IoT* protocol stack has even emerged in practice, embedding this features with an own additional *adaptation layer* located between the link and the network layer (cp. *6LoWPAN* adaptation layer in [73, Fig. 5.9]. Modeling these features and protocols for network simulation is a necessary step to accurate evaluation of wireless networks in the context of the *IoT*.

- **Routing** is one of the central tasks communication networks and thus also an integral part when modeling the network layer for wireless network simulation. The procedures for traditional *IP Wide Area Networks (WANs)* differ from those in resource-constrained wireless mesh and ad hoc networks. In [38, Sec. 16.2] a classification of mechanisms for the key modeling features (e.g., *addressing*, *topology maintenance*, and for the *path selection*) are summarized. A protocol typically is used in *WSNs* and in the *IoT* is *RPL* (cp. [2] and Figure 1.2).
- **Transmission flow** and **congestion control** on the *transport layer* are well-established with the most important internet protocol suite protocols *TCP* and *User Datagram Protocol (UDP)*. They have a major impact on the overall network performance in terms of *Round Trip Time (RTT)* and *troughput*. Especially at the transport layer, an indirect modeling approach based on real packet *traces* in contrast to correct protocol models is discussed, as trace files are easy to utilize and implement in packet-level *DES* [38, Sec. 17.4].
- **Traffic** modeling is usually assigned to the *Application Layer (APL)* and highly depends on the application area, e.g., amount of (sensor) data or burstiness of a stream transmission. Modeling is either based on algorithmic packet generation or application protocol based processing of data objects.
- **Application** paradigms and mechanisms can usually be modeled well for simulation, but depend in particular on the very diverse application areas (ref. [74, Ch. 3]) and protocols (cp. Figure 1.1 (b)). An application protocol which has particularly shaped the application layer of resource-constrained wireless networks is *CoAP* [75, Sec. 3.4]. As *CoAP* is also a fundamental part of the *Contiki* operating system, simulation based performance evaluation is often conducted with the *Cooja* (ref. Section 2.3) simulator [76], based on the native protocol implementation.

2.2.5 | Network Topology and Mobility Modeling

Modeling the network topology and mobility is substantially different for resource-constrained wireless networks compared to the traditional *IP*-based wired *WAN*. Wireless network modeling is based on geographical positions within a radio environment and must be abstracted for simulation. In the following, we briefly introduce these two modeling areas which are of major importance in this work in addition to the lower protocol layers.

Network topology modeling is a key task for evaluating wireless communications, as the underlying structure of a node network has a significant impact on the operating mode and the complexity of devices and protocols. The topology represents a certain reachability (*edges*, E) among different devices (*vertices*, V) of the network in a defined structure, which can optimally be represented by a graph $G = (V, E)$. In layered network architectures, we have to distinguish between the *physical* (link) and the *logical* (overlay) topology. The former is defined through the device connections by the physical medium, which is within the scope of this thesis.

Because the wireless channel is inherently a broadcast medium, even the physical link topology is an abstraction and must carefully be considered in modeling wireless networks. This process includes several levels of abstraction (e.g., the classification and unification of different device types into vertices, the conversion of the attenuation values between nodes into edge attributes (*weights*), or the elimination of vertices and edges according to merging network components to a single entity). Furthermore, a graph can be described with structural properties to achieve the required characteristics of a given network evaluation scenario. In network simulation modeling, topology models (generators) can be used to obtain different pseudorandom graphs depending on the desired properties (cp. topology models in [38, Sec. 22.4]).

Mobility modeling for a network is based on the link topology whose graph properties change depending on the node movement. Mobility, however, can also be considered as an effect of higher protocol layers according to the overlay topology, e.g., the *NWK* [38, Sec. 16.3], since address allocation is dynamic at this layer and thus address changes are possible without physical movement in the radio channel domain, e.g., according to network failures or logical reconfiguration. On the other hand, a *handover* due to a change of the physical location (which must then be modeled accordingly in the radio channel domain), for example, does not necessarily constitute a change of address.

Mobility effects can ultimately have an impact on the operations at the higher layers, so they should also be considered in the modeling of more complex topologies and applications, especially related to the *IoT* protocol stack. As defining physical node movement is very elaborate from scratch, mobility modeling is an important field of research with many different models and approaches, also applied in communication network simulation (cp. mobility models in [38, Sec. 14.3]).

2.3 | Selected Simulation Systems and Models

Practically all simulators for (wireless) communication networks and protocols serve the *DES* domain. This section does not aim at completeness because the number of available simulators for different research aspects is almost unmanageable and the commercial sector also plays a significant role, which is more or less neglected here. However, some simulators that can be used in academia have become established over the years and are currently being developed further, while other tools are losing relevance.

2.3.1 | Discrete-Event Simulators in the Wild

With the adoption of the *IEEE 802.15.4 WPAN* standard specification and the rise of *WSNs* in the 2000s and 2010s, specially tailored and derived network simulators or virtual testbeds with software emulation support became popular alongside sensor node operating systems (cp. [77] for a comparative analysis) and hardware platforms. Some relevant examples of this classification include

- TOSSIM [78] (a *DES* environment, simulating real code applications of the TinyOS [79] operating system with the latest release version 2.1.2 in 2012),
- Cooja [80] (a simulation facility for the Contiki [81] operating system with latest release version 3.0 in 2015, still supported with Contiki-NG[1] version 4.7 in 2021),
- AvroraZ [82] (builds upon TOSSIM with AVR[2] processor emulation for specific sensor node platforms), or
- Desvirt [83] (a testbed virtualization framework alongside the *native*[3] motes hardware virtualization of the RIOT [84] operating system).

As *WSNs* and the *IoT* become commercial in the last decade, these tools are play only a minor role according to deprecated hardware, use-spread or their software specificity, when looking at modern system requirements and widespread hardware platforms. Nevertheless, current state-of-the-art simulation software for wireless communication networks and protocols without node hardware or platform dependencies is also divers. Its representatives can be found, e.g., in [85], [86], [43], [87], [88], and [89]. In Table 2.2 some recommended software for the simulation and emulation of wireless networks is shown. The selection in [88] is based on several requirements in addition to wireless simulation capabilities, e.g., low investment, suitability for higher education, ease of use, features covered, popularity, skills required by users, and evidence that it is a vibrant and active project.

[1] Contiki-NG designates the successor development of the Contiki operating system for resource-constrained devices in the *IoT*: `https://www.contiki-ng.org/`

[2] AVR is a family of microcontrollers from the Microchip Technology company.

[3] *RIOT native* corresponds to a hardware virtualizer allows the compilation and execution of wireless sensor node applications as a user process on the host operating system, e.g., Linux.

	Packet Tracer	**Mininet**	**ns-3**	**OMNeT++**	**CORE**
Mobility	*	****	****	****	****
Handover	*	****	****	****	****
Device Config	*****	**	*	*	*
Radio Packets	**	*****	*****	*****	*
Signal Range	*	***	****	****	****
Interference	*	****	****	****	***
IoT	***	****	****	****	*

Table 2.2: Comparison of recommended simulation and emulation systems based on usage criteria in the context of wireless networks (excerpt taken from [88], whereby pure *IEEE* 802.11 *Wireless Local Area Network (WLAN)* relevant features and systems are eliminated).

Whereas *Mininet* (ref. Table 2.2) is a still very young project, predominantly featuring *IEEE* 802.11 features (Mininet-Wifi [90]), the analysis in [88] shows that the most prominent simulators *OMNeT++* [91] and *ns-2/ns-3* [92] do not differ significantly in the selected properties for wireless network simulation (according to *WPANs*, *WSNs*, and the *IoT*). Furthermore, analyzes in [87] have demonstrated that they are capable of carrying out large-scale and efficient simulations, while *OMNeT++* can be considered as viable in terms of availability of wireless communication models. As this thesis prototype is build upon *OMNeT++*, a brief overview is given in the Appendix B.

2.3.2 | Link Layer Simulation Models and Systems

This section gives an overview of models and systems with relevance to wireless link layer simulation, besides model frameworks coming with the simulation tools (e.g., *OMNeT++* models[4] or ns-3 models[5]). In the following, we give an overview and present selected tools with different approaches for link layer modeling of wireless systems in the field of *WSN* and *IoT*. Furthermore, our *OMNeT++*/INET *IEEE 802.15.4* simulation model is introduced in detail, which provides due to its modeling accuracy a decidedly valuable basis for the objectives in this thesis.

In [70], E. Municio et al. present a tool for simulating *6TiSCH* networks to estimate the performance according to the *6TiSCH* specification (cp. Figure 1.2 (c) in Section 1.1 for a stack reference and *RFC* 8180 [33] for the specification). It focuses on simulating the network behavior that can be observed from the *MAC*, e.g., accurately monitoring the network formation, routing, and scheduling. Since the *PHY* abstraction level is based

[4] *OMNeT++* Simulation Models and Tools: `https://omnetpp.org/download/models-and-tools.html`
[5] ns-3 Model Library: `https://www.nsnam.org/docs/models/html/index.html`

on the *Time Slotted Channel Hopping (TSCH)* slot time quantization, modeling the *PHY* features (ref. Subsection 3.2.3, e.g., *ED*, *Modulation*, etc.) or standard-specific services and general radio specifications (cp. [9, pp. 27ff]) is completely omitted. A further abstraction was introduced with the protocol message objects that carry only relevant parameters without regarding the accurate message bytes. The higher layers (from the *MAC* and above) with its protocols (e.g., *TSCH*, *6top*, *IPv6*, etc.), are highly configurable. Besides a variable, generic traffic scheduling, specific models for transport or application protocols were not implemented in the simulator.

Simulating the *IEEE 802.15.4*e [24] *Deterministic and Synchronous Multi-channel Extension (DSME) MAC* is presented by F. Kauer et al. with the *OMNeT++* simulator-integration of the research platform *openDSME* [48, pp. 451ff]. It has been the first open-source implementation of the *DSME MAC* specification (cp. [48, pp. 454ff] for an explanation) and as such it can run on real sensor nodes with *Contiki* and *CometOS*[6] [93] host operating systems. The sensor nodes *DSME* stack layers provided with the platform, are integrated as a monolithic *DLL* module in *OMNeT++*/INET via the `IMACInterface`. Thus, the communication between modules inside the *DSME* protocol implementation is not traceable with *OMNeT++* messages. As a *DLL* in the simulator, on the other hand, openDSME communicates with the *NWK* and the *PHY* via message events, while *OMNeT++*/INET provides the upper and lower layers. While the *SAPs* to the upper layer are modeled according to the standard specification, the interfacing to the *Physical Layer Management Entity (PLME)* of *IEEE 802.15.4* is omitted, and the inter-layer communication is realized by means of the *DSMEPlatform*.

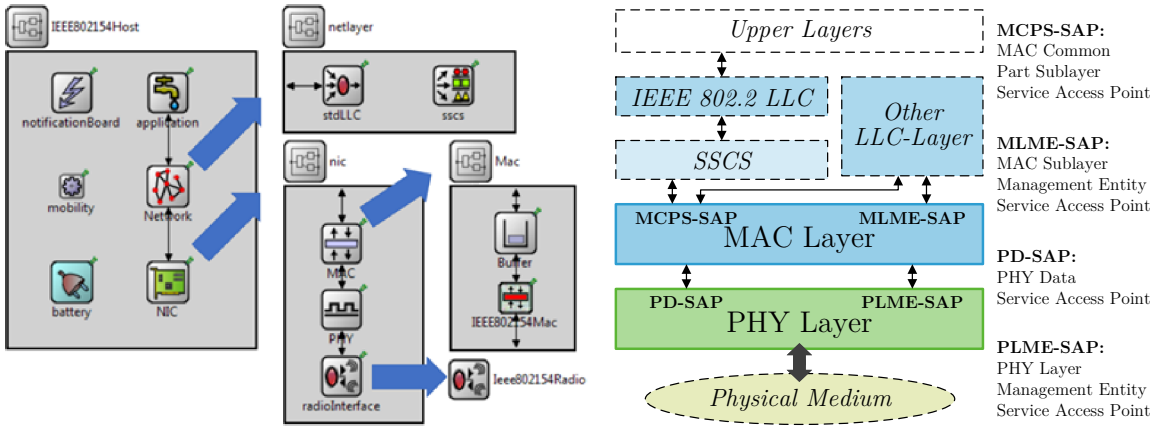
M. Slabicki et al. have introduced *FLoRa*[7] (Framework for LoRa) [94], a framework to enable end-to-end simulations of *LoRa* networks in *OMNeT++*/INET. The LoRa networks technology relies on two components that correspond to different layers of the protocol stack: LoRa *PHY* and *Long Range (LoRaWAN)* open specification for the *MAC* protocol and *NWK* layers (cp. [94, Fig. 1] for the LoRa protocol stack and the available simulation modules). For the *PHY* model, the transmission parameters, e.g., spreading factor, center frequency, bandwidth, code rate, and transmission power, can be configured to determine the communication range and the occurrence of collisions. To support urban and sub-urban radio environments, the model is parametrized with real measurements from a parameter study [95] in corresponding areas. Furthermore, FLoRa supports transmission interference based on a model that assumes collision occurs in non-orthogonal channels when two messages overlap in time, but, with a sufficient power difference (assumption: more than 6 dBm) between two colliding signals, the stronger one is decoded due to the *capture effect* [94]. However, the *PHY* model is solely based on a parametrized path-loss model with shadowing (cp. [68, Sec. 2.8]) in which no radio modulation characteristics, interfaces, services, or features are modeled.

[6] CometOS: A component-based, extensible, tiny operating system for wireless networks, highly similar to *OMNeT++*'s communication paradigm: `https://www.ti5.tuhh.de/research/sensornet/cometos/`

[7] The *FLoRa* framework project page: `https://flora.aalto.fi/`

OMNeT++/INET IEEE 802.15.4

In [96], we introduced a new simulation model for the popular *IoT* and *WSN* communication standard *IEEE 802.15.4* [9]. This *OMNeT++*/INET 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 (ref. protocol stack reference in Figure 2.7b) and general modeling guidelines for 802.15.4 [97], close to the formal protocol layer model in Section 2.2. In Figure 2.7a an exemplary *IEEE 802.15.4* host with its *submodules* is shown.



(a) The exemplary simulation host with its submodules (taken from [96])

(b) The *IEEE 802.15.4* protocol stack reference according to [9]

Figure 2.7: Block diagram of the *IEEE 802.15.4* model in *OMNeT++*/INET compared to the protocol standard specification.

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, whereby the packet reception and transmission is modeled according to the service primitives as specified in the standard. The *MPDU* packet definitions include all types and fields that are specified in the *IEEE 802.15.4* standard [9]. All the modules are initialized with parameters (either set to default values or specified by he user in the simulation configuration) according to the *OMNeT++* stage initialization procedure. The model also includes prepared use cases in different operation modes (e.g., direct data transfer, indirect data transfer, *GTS* handling) as example configurations. Depending on the *MAC* operating mode to be simulated (cp. [9, pp. 67]), different timers (e.g., the *Beacon Interval (BI)* in *Beacon*-enabled *PAN*, or the receiver state change of the radio module) are started internally with the module initialization. For the higher layer simulation, the *6LoWPAN* model [72], for instance, can be set up to work with.

2.4 | Limitations of pure Simulations

In this section, we conclude the analysis of modeling and simulation for resource-constraint wireless communication networks by showing the limitations of purely virtual simulations. There are a few surveys about network simulation limitations in general in [37, 47, 59, 98] and sprawling specific discussions about limiting factors by utilizing a concrete model. The analysis in this section provides a specific overview of the limitations based on this thesis's focus on simulating the wireless link layer within an evaluation of holistic wireless networks from the protocol stacks point of view. The most notable limitations can be classified into the categories presented below in the appropriate subsections.

2.4.1 | Validation and Credibility

Important limiting factors of wireless network evaluation with pure simulations are the validation and credibility of simulation results. The validation of simulation results is usually based on comparisons with field tests from an input-output transformation point of view, but they only consider a selected set of parameters which is not sufficient to evaluate the quality of a model. For conformance tests and validation, a solution is to let all simulation models and tools pass through an independent test regarding their standard specifications provided either by an *RFC*, *ETSI*, *ITU-T*, *IEEE*, or proprietary specification (ref. wireless network link layer protocols in Section 1.1). Appropriate *Independent Verification and Validation (IV&V)* attempts have rarely been undertaken in the network simulation community [98] so far.

Developing simulation models for a specific purpose or application is still the prevailing opinion [38, Sec. 1.2], [74, Sec. 6.2], [39, Sec. 1.3] which leads to over-simplistic models, a reduced reusability, and, in the end, a generally large model variety. Especially in the *OMNeT++* ecosystem, a strong community with very many simulation frameworks has been formed in the last decade (cp. [48, Ch. 2]). This is at the same time a drawback, because most of the models are developed for a certain purpose and abstract or neglect other features (cp. analyzed models in Subsection 2.3.2). They are unsuitable for other research questions and far away from being universally valid. Therefore, we propose to pay attention to the exact and holistic protocol modeling according to its specification, especially in the architecture of the model and an easy parametrization according to the respective requirements (cp. [48, pp. 451ff] for the simulation model of the *DSME MAC*). Furthermore, our *IEEE 802.15.4* simulation model (ref. Subsection 2.3.2) for *OMNeT++*/INET claim to satisfy the requirements of the *PHY* protocol architecture, features, and interfacing. In the result of the other analyzed link layer models, there is no simulation model that tries to exactly represent the *PHY* protocol architecture, neither in functional scope nor in structure of modeling.

Protocol simulation studies are often conducted to show the algorithmic effectiveness

of selected protocol schemes and features by abstracting or omitting several details [98]. For example, the model of the *IEEE 802.15.4 GTS* simulation study in [99] has several limitations regarding *MAC* features and abstracts completely from the *PHY* details. The modeling features (Section 2.2) and the state-of-the-art simulation models for wireless link layer protocols (Subsection 2.3.2) indicate the increasing abstraction, when modeling the lower layers of a wireless transmission system. It is also noticeable that often when *PHY* models are mentioned the signal propagation is actually meant (e.g., the LoRa *PHY* model in [94] or the *6TiSCH* simulator *PHY* model in [70]). But any ignoring of protocol parameters and properties from the specification will cause the results to deviate from reality. Almost all physical layer models belong to the frame or packet domain and assume that the most *PHY* functions work perfectly without any performance degradation [38, p. 157]. There is practically no performance simulation study that involves signal modulation. In contrast, most simulation studies make highly simplified assumptions about the physical conditions which provides little confidence in the credibility of the simulation results.

Regarding a concrete specification, such as the *IEEE 802.15.4*, there is in addition many room for interpretation or explicit hardware dependance within the standard definition, e.g., a concrete parameter setting of a radio module is often left to the manufacturer. For *IEEE 802.15.4*, this leads to differences in the calculation of *GTS* time slots, *ED* range, energy consumption, and execution times. To mitigate these issues in pure simulations, system designers would have to need explicit chip models to cover special value ranges and options. Summing up, it can be stated that accurate environmental and radio channel modeling is exceedingly unfeasible in discrete network and protocol simulation. While models of the *PHY* mostly surely represent the data transmission and reception, this assures that simulation alone is not the first tool of choice in the field of wireless network performance evaluation. This is also reflected by the current trend that the majority of evaluations are conducted using experimental studies in testbeds and that this number has been increasing slightly in recent years [37].

2.4.2 | Performance and Scalability

Some obvious advantages of network simulators, e.g., high scalability, are only valid when purely virtual evaluations are performed in which simulation time is not an issue. However, if the algorithmic complexity of the simulation models and thus the simulation time increases enormously, studies can become unmanageably time-consuming and exclude the coupling with external interfaces in a *Real-Time DES* setup (ref. Section 2.1.3). In [100], E. Egea-Lopez et al. surveyed the main scalability issues in *WSNs* simulation and showed both, the need for integrating detailed and accurate physical models, and the crucial impact on scalability (both network load and number of nodes) with partly exponentially degraded performance (cp. the performance results of the full model in [100, Tab. 1]). A similar effect (albeit only with quadratic complexity) can already be

observed in simulations without considering exact physical transmission models, e.g., the 6TiSCH *MAC* simulator [70] presented in [70].

These problems are further amplified when other model-relevant aspects in the context of *WSNs* (e.g., node mobility, energy consumption, or sensors) are added to the protocol-related modeling. Managing these performance and scalability issues eventually leads to sacrificing accuracy by partitioning and optimizing the network model and by using lightweight mathematical abstractions [100]. Parallelizing computations with *Parallel Discrete Event Simulation (PDES)* setups can also possibly reduce this problem, but there are still a lot of open issues in network simulation for future research [101], so sequential simulations remain the norm.

2.4.3 | Cross-Protocol Stack and Interference

Most of the simulators and models are designed for evaluating the performance of protocol algorithms in homogeneous protocol architectures and networks. For example, with the INET framework, the *OMNeT++* community has created a basis for modeling heterogeneous networks by means of different host types and protocol stacks. When looking at the protocol variety of the wireless link layer (which is more complex to model in *DES*, but indeed responsible for the physical data transmission), there is still some potential for model development. In fact, many existing simulation models of wired broadcast networks ignore collisions and interference completely by loosing only little accuracy [64, pp. 27ff] because of the well-known collision detection mechanisms and the exactly predictable signal degradation on a wired medium. However, this cannot be applied to wireless networks, since the effects of collisions and interference are significantly influenced dynamically.

Simulation models describing the effects of *Cross-Technology Interference (CTI)* are practically non-existent. If *CTI* is considered in network simulation it is usually based on data sets from real-world experimental measurements (e.g., [102]). The few existing physical *Inter-Channel Interference (ICI)* models greatly simplify [103] and are not used in practice. Finally, simulating protocol interference and physical interference (ref. Subsection 2.2.1) at the wireless link layer is unsuitable in *DES*. Also the review in [104] shows this little significance in network simulation. Studies of cooperating wireless link-layer technologies, as well as the *ICI*, are obviously still the domain of experimental evaluation in field tests.

3 | Wireless Network Emulation

Realistic evaluation plays a crucial role in commercial and academic development of network devices, communication software, and applications. In contrast to academic research which focuses more on simulation for the evaluation of algorithms or protocol procedures, industrial development is mainly characterized by often extensive and costly device testing in real or *emulated* networks, e.g., in addition to in-house test departments in companies developing and manufacturing wireless communication systems. For this, there are dedicated specialists for modern wireless test procedures. Nevertheless, network emulation is a broadly established and fundamental evaluation approach with a beneficial tradeoff when considering *experimentation cost*, *reliability* of the results, and *flexibility* (cp. [1, pp. 16ff], cp. [38, pp. 83ff], [36], [105] for a detailed overview of this area).

In this chapter, we summarize important basics and emerging techniques for emulation-based wireless communication network evaluation. In addition to basic requirements and strategies, we present several approaches in overview and comparison, as well as selected concrete systems in detail. Accordingly, the *RoSeNet* system (ref. Section 3.3) is considered exemplary as a hardware platform on which the prototype for evaluation in the context of this work is built. Based on the analytical overview of related emulation techniques, systems, and approaches, this chapter concludes with a discussion about the limitations of network emulation to underline the demand for hybrid evaluation techniques in general and the *Radio-in-the-Loop* concept and prototype introduced with this thesis in particular.

3.1 | Emulation Methodology and Classification

Network emulation aims at creating an environment for interconnecting real-world components (e.g., devices, applications, services) being tested in a controlled laboratory environment with regard to network conditions [1, p. 14]. The emulation technique often uses both physically real network hardware and virtual components, e.g., based on computer models. Thus, network emulation approaches can increase the accuracy and reliability of evaluation results in comparison to the simulation. They help to answer questions like: *What performance can be achieved by the implementation of a particular protocol, application, or transceiver device architecture in a wireless network?* Since the field of network emulation is sometimes very vaguely defined in the literature and used in different contexts in practice, a methodical delimitation and a concrete classification is presented in the following for this thesis.

3.1.1 | Methodical Delimitation

The basic methodology aims to produce the network conditions for a test scenario as realistic and accurately as possible to obtain reproducible and detailed feedback from the designed system, but without modeling all components exactly (compared to simulation). Thus, emulation can be described as an imitation of a real-world process based on "something such that it is equivalent to the original entity" [39, p. 3]. The application domains of network emulation in the field of wireless communication systems are the lower protocol layers, topology and mobility, and device hardware. Appropriate approaches and architectures can also usefully extend or combine existing evaluation concepts, such as simulations (e.g., network emulation in *OMNeT++*/INET [48, pp. 88f]) and testbeds (e.g., *Network Emulation Testbed (NET)* [1, p. 205] and *Virtual Testbed (VTB)* [106]), to enable more accurate investigation results. The emulation of the device hardware (e.g., processors, control units, or the sensors/actuators) is explicitly not considered in this thesis.

As the network emulation methodology does not necessarily focus on models, the terminology can be different from simulation modeling. Basic terms like *entity*, *system*, and *attributes* are used in the same context. However, when not considering models as a basis for the network emulation setup, a common understanding is given with the generic term *black box*. Thus for network emulation, only the inputs and outputs are important and not the internals. Nevertheless, emulation systems are built up from individual building blocks (modules) to enable configurability and reusability, and to ensure certain properties. For example, the link quality degradation ΔQ [1, p. 9] can be an important property of the black box system for a network link emulation. Thus, a network link emulator appears to be a real wireless communication link to which physical real end-systems can be attached.

3.1.2 | Classification

Since network emulation covers a large area between model-based evaluation and real component tests, we summarize possibilities for classifying network emulation systems in specific categories in the following. Based on the architectural composition of network emulation systems using real and virtual components, a subdivision into hardware- and software-based techniques is obvious. The principle of *real hardware*-based emulation involves the use of hardware components to statically specify and influence network properties or to decouple time-critical functions and computations from complex models. Especially on the *PHY* and the radio channel domain, where the involved hardware is specifically adapted to the network technology, this primarily supports the repeatability of tests under reproducible conditions. For example, analog or digital hardware components can emulate the radio link via parameterizable link properties, or parts of the device hardware and network interfaces can represent a transceiver.

(Real) *software*-based emulation also serves the purpose of influencing network properties in a controlled manner, such concepts though are mainly considered beyond physical transmission. Thus, the execution of real communication software can be embedded or used as a host operating system service, e.g., to avoid abstract and inaccurate modeling. This can be performed both on the general purpose processor of the control or runtime system, and with the help of special instruction-level processor emulation systems. *Virtual* software should be mentioned in relation to model-based emulation which requires no special hardware. These components are quite versatile. They can, for example, enable artificial delays or losses during a real packet transmission.

Nevertheless, the concrete implementation of all components for emulation depends decisively on their purpose and classification within the communication network. A fundamental classification (derived from [1, p. 270]) following the protocol layer models (ref. Figure 1.1) is presented below in Table 3.1. The classes allow an beneficial characterization and assignment of approaches and systems of network emulation with the delimitation areas, the testbed, and the simulation (*class D* corresponds to the simulation when all layers are pure *model-based*). Based on the classification keywords, the layer

	Testbed	**Class A**	**Class B**	**Class C**	***Class D***
Applications	*real*	real	real	real	*emulated*
Protocols	*real*	real	real	emulated	*emulated*
Interfaces	*real*	real	emulated	emulated	*emulated*
Conditions	*real*	emulated	emulated	emulated	*emulated*

Table 3.1: Classification of network emulation systems (derived from [1, p. 270])

structure from the physical radio environment via the network interface to the network application is apparent: *Network applications* map services, (distributed) applications and tasks, and application data flows in the node network. *Network protocols* map the functional specification, from transport to application layer protocols (*APL*). *Network interfaces* refer to the link layer with the associated hardware (e.g., digital and analog signal processing in the circuits) to access the radio channel (cp. modeling domains of the *PHY* in Figure 2.6). *Network conditions* represent the effects of signal transmission at the *analog radio domain*, according to Figure 2.6 in Section 2.2.

As already mentioned in the previous chapters, the performance evaluation is highly effected by the link layer (cp. the internet protocol suite in Section 1.1). It is therefore not surprising that in the field of wired and wireless network emulation approaches are mainly present at the lower layers of the protocol stack. The classification in Table 3.1 also highlights that emulation of the physical connection (network interfaces and conditions) is the most important component of network emulation systems, as it is involved regardless of the system class. The emulation of the physical connection (*radio link*) is considered as a basis in this thesis context and is presented hereafter.

3.2 | Radio Link Emulation Requirements and Strategies

Radio link emulation covers the network interfaces and radio conditions in the network environment (ref. Table 3.1). The emulation domains for *applications* and higher layer *protocols* are omitted in radio link emulation. A link-level emulator (cp. [1, pp. 56f]) recreates the end-to-end characteristics, often expressed as network quality degradation of a connection between devices (according to the effects of wireless link congestion in Figure 2.3). In the following, basic requirements for emulation systems are explained in order to subsequently indicate various emulation strategies for network conditions and interfaces.

3.2.1 | Emulator Requirements

The requirements for network evaluation techniques (cp. [1, pp. 16ff] and [38, p. 85]) in general and the emulation in particular are summarized briefly here. We consider some of them as basis in this context. They should be fulfilled from all radio link emulation systems:

- **Repeatability** of an evaluation scenario is usually one of the main objectives and a basic requirement. Therefore, the system must have the full *control* over experimental conditions to *reproduce* the evaluation results. A distinction can be made between centralized and distributed approaches for *emulation control* (cp. [1, pp. 64ff]). For the centralized emulation control, e.g., based on a single control unit, the processing of the configuration commands (both node and network components interaction) can become a bottleneck with an increasing network size.
- **Real-time capability** is usually an important requirement for various emulation strategies. Depending on the system characteristics and the objective, this usually plays a superordinate role when connecting and running real-world components, e.g., a precise timing of *MAC* algorithms, the communication with real chip hardware, or the signal transmission within a real fading channel. This implies, however, that model-based components also comply with the requirements for real-time execution and are accordingly ready with their calculations on time.

Furthermore, some more requirements are discussed in addition which should be met to a desired degree depending on the evaluation goal. This parameter selection and differentiation serves at the same time the comparability of different approaches analyzed later in Section 3.3:

- **Scalability** at a high level is necessary to show how the network behaves and performs for a given network size, as well as to figure out the limitations of the network size, especially in wireless networks with shared medium communication. The parameters of scalable wireless networks are, for instance, the *number of*

nodes, the *amount of traffic*, the *number of parallel networks* according to channel multiplex, or the *number of hops* in a *mesh network*. Because commonly designs for emulation testbeds or *Device Under Test (DUT)* evaluation scenarios are limited to several single nodes, evaluating large-scale deployments is often unfeasible. According to Beuran [1, p. 256], the size of an emulated network is limited to tens of nodes in centralized control solutions. Adding computing resources in distributed and parallel approaches can increase the scalability and overall system performance, but they are very complex to implement.

- **Mobility** refers to the capability to emulate physical node movement. As there are several techniques to achieve mobility, presented in the following section, this requirement may vary in accuracy or may be absent entirely. However, a small, imprecise degree of mobility is always present when it is possible to switch nodes or their radio links on and off during emulation. In terms of modeling node position changes in a scenario, simplicity and the use of models, such as motion patterns, is crucial.
- **Modularity** is a property that goes hand in hand with configurability and interchangeability, which is difficult to achieve with an inadequately structured architecture. It should therefore be consistently layer- and module-based, easy to extend, and not a monolithic black box for a single application. What is the standard for model-based simulation (e.g., model libraries or community frameworks) is partly not very common in various emulation techniques. It is not uncommon for emulation systems to have a strong proximity to application-specific *testbeds*.
- **Configurability** refers to the flexibility to adapt the emulation setup for a certain research question. Because an emulation system that can only represent a single scenario is more likely to be considered a *field test* with emulated properties. For this reason, why it must shielding the complexity and abstract from hardware and system details by offering adjustable levels of the above mentioned emulation features. Usually, a *Hardware Abstraction Layer (HAL)* [38, p. 88] is considered in the system architecture to reduce the complexity of the configuration. It is also beneficial to offer (external) tools to automate the scenario configuration, e.g., by the use of a *GUI*. Since the network emulation is an *abstraction* of a real network, its *modules* should be configurable by a parameterizable model representation. Due to diverse dependencies (e.g., regarding real hardware components or system limitations), low flexibility often causes the biggest limitations in the wide use of emulation systems [1, p. 256f].
- **Traceability** is also important, as it enables further processing and evaluation of the achieved emulation results. What is desired, is a possibility to record nodes' packet data exchange as a trace, a monitoring of component events and execution processes, as well as but not even, a selection of emulation result parameters. This is certainly reflected in the degree of traceability. Simple evaluation data, such

as data rate and bandwidth, might be insufficient for obtaining insights into the emulation process.

- **Accuracy** is generally really good (cp. [38, Fig. 6.2, p. 85]), because by using real components, the overall system is usually closer to reality than with purely model-based methods. With regard to the emulation of the radio link, however, a distinction and quantization must be made according to several aspects, e.g., on the one hand, realistically modulated packet data including transmission by means of real wireless hardware and, on the other hand, purely virtually generated packet data (sometimes with attached frame meta data) and a transmission on a wired testbed network. In addition, the accuracy of radio transmission effects, e.g., interference and noise, plays a significant role compared to a pure consideration of the device under test transmission signals which can ultimately have a strong impact on the accuracy of an evaluation question.

- **Isolation** for hardware-based approaches with real interfaces means the protection against radio pollution and disturbances. We can distinguish *internal isolation*, on the one hand, to prevent undesired multipath effects and avoid signal leakage and spreading via control network and, on the other hand, *external isolation* to shield the internal *Radio Frequency (RF)* architecture and components from interferences [107, p. 98ff]. In purely model-based/virtual emulation approaches, of course, this physical isolation does not matter, since no real packet transmissions are triggered. In this context, isolation means protection against other effects that influence the transmission parameters or computing performance of the overall system. For example, if the emulation system is based on a testbed infrastructure, it must be ensured that transmission power and resources on this network system are sufficient for all relevant operations and that these are not disturbed by other data transmissions (e.g., higher priority protocols, management, cyclic control, etc.).

3.2.2 | Emulating Network Conditions

In simulation systems signal propagation effects are usually abstracted through virtual packet data modification. When considering radio link emulation, the physical packet transmission with configurable network conditions is recreated based on different strategies. Solely emulating the radio environment (in *class A* emulation systems) results in a single discipline, called *radio channel emulation* or *fading simulation* (when using digital hardware). The emulation of network conditions is often applied only to a sender-receiver relationship, as depicted in Figure 3.1. We wdistinguish here between the emulation of radio channel effects of a single network link, the radio topology of the network, and the mobility of the physical nodes. An overview analysis of concrete systems strategies is presented in Section 3.3.

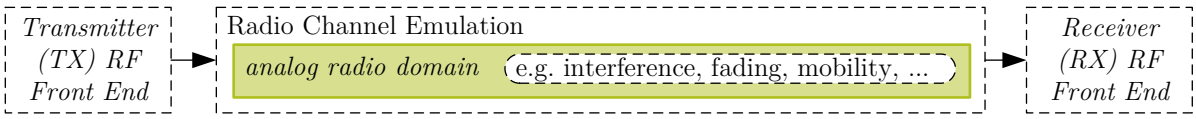


Figure 3.1: Radio channel emulation of wireless transmission systems for sender and receiver according to the *analog radio domain* in Figure 2.6

Radio Channel Emulation

Radio Channel Emulation offers a way for recreating realistic *RF* communication conditions. Beuran [1, pp. 264ff] distinguishes between *real* network conditions for wired network emulation, which can be either *uncontrolled* or *controlled*, and the use of computation models for *virtual* emulation. This classification does not fit exactly for wireless networks. *Uncontrolled* real conditions arise when using real components, as much as possible, in a real environment, which increases the result accuracy, but can be regarded more as a *field test* rather than an emulation with a limited ability to reproduce certain conditions. Pure software-based (*virtual*) emulation is build up on modular software components, whereby, in contrast to simulators, no exact model is reproduced for simulation, but instead existing infrastructure (for a purpose other than the intended one) is used (e.g., emulating wireless transmissions over wired networks).

In order to provide the system user with a *controlled* influence on the real conditions (e.g., for repeatability with modifications), the use of configurable system components is indispensable. This is achieved in practice by using dedicated emulation hardware (*analog* or *digital*) to capture and process real radio signals. The emulation hardware components can then be configured to affect the channel quality parameters presented in Subsection 2.2.1 The wireless devices are isolated from each other and connected over their radio interfaces to the *RF* channel emulator hardware, which emulates signal propagation effects, as already presented in Subsection 2.2.1. 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 which support hundreds of interconnected nodes. Nevertheless, applications run as static firmware implementations on real-life wireless network hardware and flexible adjustments of applications or protocol parameters are complicated compared to network simulators.

Network Topology and Mobility Emulation

The most of the modern link-level emulators attempt to achieve a realistic or high accuracy end-to-end connection with a complex setup. Thereby often only two systems (sender and receiver) are related to each other. Advanced approaches also allow the emulation of a whole network *topology*, so that connections of several different nodes and

scalable networks are taken into account, such as topology-level emulators [1, pp. 57ff] different device types (device topology), and equal devices in a network eventually resulting in a certain radio topology. What is still relatively easy to implement for a single transmitter-receiver system becomes highly complex for larger topologies and complicated environmental conditions (especially in industrial environments, indoors), e.g., the setting of appropriate *attenuators* in analog hardware emulation is done on the basis of the model-based attenuation value calculation.

Mobility is a very important feature when looking at wireless network modeling in general (ref. Subsection 2.2.5) and channel emulation in particular, because it introduces special network situations (e.g., reaction to a sudden decrease in signal strength). This can be either based on robots, moving a node around in a controlled environment, *antenna switching* approaches [105, Sec. 3.2] or variable step attenuators (e.g., with an attenuation matrix in [108]), which set the appropriate values for long-term fading emulation. The movement of nodes in a radio channel results in a constantly changing radio topology. For this purpose, mobility models must be developed that permanently generate technology parameters, such as attenuator values, from changing node positions.

3.2.3 | Emulating the Network Interfaces

When looking at class A emulation systems, usually real network devices and interfaces are used in the evaluation setup. What is very suitable for device testing can no longer be applied when complete control over the behavior of the interface is required (e.g., flexibility of interface-to-channel interaction). The emulation of network interfaces is mostly based on abstract parameters (e.g., bandwidth, *PER*, or operating channel frequency) comparable to the abstraction in simulations. When incorporating a detailed view on more realistic strategies, from our point of view, the emulation of wireless interfaces must be divided into the radio front end and the specifics of the network interface protocols at the link layer (mainly *MAC* and *PHY* features).

Radio Front End Emulation

The emulation of the radio front end is assigned to the nodes signal processing or the *waveform domain* when considering the decomposition into modeling domains of wireless communication systems (ref. Figure 2.6 in Section 2.2). This area includes components where the signal processing for the generation of radio packets can flexibly be adapted. This can be achieved on the one hand with highly configurable *Digital Signal Processor (DSP)* and *Field Programmable Gate Arrays (FPGAs)* and on the other hand based on *SDR* concepts on general purpose hardware. Although *SDR* techniques are not ubiquitously assigned to emulation, concepts in which signal processing (i.e., on hardware devices) is modeled on general purpose hardware formally meet this definition

and can often be found in the literature (e.g., [109]). Pure software-based transceiver modeling can be used to enable realistic but flexible and configurable *PHY* transmissions. In Figure 3.2 the signal processing of a *PPDU* via an generic *PHY SDR* transmitter (e.g., *QPSK* modulation) to the antenna interface is depicted.

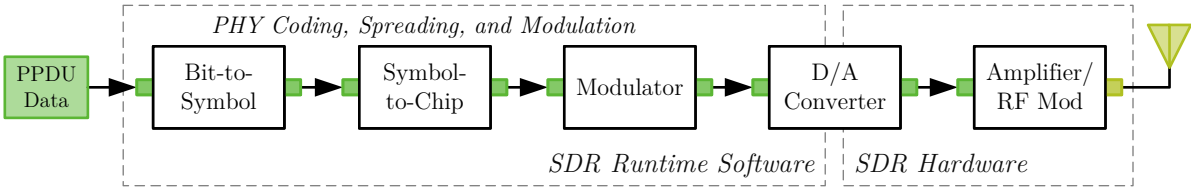


Figure 3.2: Example PHY SDR Transmitter

The concept of *real-time streaming* is the basis for *SDR* concepts at large and a very active area of research, but not state-of-the-art in radio module development. It is very complex to develop a complete *SDR* transceiver front end and, in general, the designs are more energy-intensive and costly as single-chip solutions. Other limitations of *SDR* concepts can be studied in [110, Ch. 4]. On the other hand the deployment of flexible wireless network evaluation can be very beneficial, when getting along with the performance issues in scalable network scenarios.

Link Layer Protocol Emulation

According to the *TCP/IP* protocol stack, the network interface (link layer) comprises *MAC* and *PHY* features. The *PHY* (layer 1) architecture or service procedures are often not of any interest because they increase the system complexity without a clear accuracy gain. However, to access and configure platform-depended components in a standardized manner or to feed physical real signal processing with virtual or even simulated higher-layer protocol data an accurate architectural implementation of the protocol interfaces can dramatically increase the flexibility of the emulation. Integrating (parts of) real protocol implementations, e.g., from open-source operating systems, is one of the key strategies for *protocol emulation* of the layers 2 and above (ref. emulation approaches in Section 3.3). This method is sometimes not clearly delineated as emulation in the simulation literature, e.g., it appears in the field of simulation as *NSC* [38, p. 88].

It is, however, also possible to setup alternative strategies to emulate protocol functions, some of which can be used in the emulation of higher-layer protocols, but also regarding the link layer. For example, the *MAC* can virtually be emulated by recreating several protocol properties (e.g., traffic patterns, frame delay, and frame drop) based on black box modules, which outwardly deliver exactly the desired properties. According to an *IEEE* 802.15.4 beacon-enabled *PAN*, this can mean to schedule a beacon frame every x milliseconds with a certain transmission probability using any data generator.

3.3 | Selected Radio Link Emulation Approaches

In addition to the methodological classification of network emulation systems presented at the beginning of this chapter, the characteristics of the technical implementation (with hardware or pure software components) can become a decisive differentiator. The exclusive emulation of the network conditions with real physical interfaces results in an emulation system of class *Class A* (ref. Table 3.1, in practice called *Channel Emulator*) with *analog* or *digital* hardware channel emulation. *Class B* approaches aim to increase the flexibility of interface-to-channel interaction and are either purely *virtual*, or use *model-based* calculations like in simulations.

Channel Emulation (Class A)		***Class B***	
analog	*digital*	*virtual*	*model-based*
Jung and Ingram [111]	*Beshay et al.* [112]	*Beuran et al.* [113]	*Elsner et al.* [114]
RoSeNet [115]	Borries et al. [116]	Wu et al. [117]	Flynn et al. [118]
Onishi et al. [119]	Matai et al. [120]		

Table 3.2: Radio link emulation systems and approaches overview, according to their class and technical implementation

In contrast to simulators and their models, network emulation systems are rather rare. They cannot always be classified exactly or follow a very different approach sometimes. Table 3.2 lists the most important research approaches for radio link emulation regarding scientific relevance and comparability, distinguished by their class and technical implementation (apart from a very few commercial *channel emulators*). Emulation approaches of the higher classes (e.g., Comer et al. [121], or Nasreddine et al. [122]) also emulate the network interfaces and conditions, but they do not introduce further concepts for radio link emulation and are not considered in detail in this analysis. On the other hand, the highlighted approaches in Table 3.2 are explained and compared in more detail below to illustrate the methodological diversity.

Class A small-scale testbed-like network setups (e.g., [111] and [119]) and, rarely to find, large-scale test solutions (e.g., *RoSeNet* [115]) can be realized with *analog* radio hardware. Since it can become very complex to enable high configurability with analog components, these approaches usually focus on a few aspects of the evaluation. Jung and Ingram [111], for instance, propose a highly accurate channel emulation testbed for evaluating *cooperative transmissions* of real nodes wireless communication with an adjustable fading channel. For another research problem, this exemplary testbed hardware connection of nodes and *RF* components (cp. Fig. 2 in [111]) is unsuitable. An excellent approach for scalable analog channel emulation is offered by the emulation system *RoSeNet*, which we introduce separately in Section 3.3 and which at the same time forms the basis for the prototype implementation of some approaches developed in this thesis.

Digital channel emulation is usually based on *DSP* with configurable *FPGAs* hardware (e.g., [112], [116], and [120]). In [112], for instance, Beshay et al. present *WiNeTesTer*, a *FPGA*-based digital channel emulator for real wireless devices testing. At the channel emulation core, the modulated analog transmission signals from the *DUTs* are digitized, processed in the digital domain, and converted back to analog for the receiving site. Within the digital processing, attenuation and amplification can be applied to emulate fading and multipath effects. Due to the modular architecture of this approach, the authors claim to enable a high scalability with minimizing the hardware resources. The general goal of both analog and digital systems is to test individual radio modules or the network performance of multiple nodes under controllable channel conditions.

Class B offers primarily pure *virtual*, emulation of interfaces and conditions. Selected approaches in this class range from model-based radio link emulation on real wired testbed networks (e.g., [113] and [117]) to pure virtual testbeds for *SDR* terminals in [114]. A *virtual channel* in VMNet [117], for instance, is based on software components manipulating *UDP* packet transmissions over a dedicated *Local Area Network (LAN)*. The parameterization of individual software modules (e.g., delay module, collision module, and bit-error module) enables a defined influence on the propagation characteristics. Thus, both this and the following approach use a standard wired *TCP/IP* network as the test infrastructure for radio link emulation.

Beuran et al. [113] introduce a combined approach for network, processor, and sensor emulation as an extension of a wired network testbed. The radio link emulation is based on the transmission of modified *PDUs* by a channel model library. They use processor emulation to generate the *IEEE 802.15.4 PHY* compliant radio packets at the network interface and reproduce a the channel condition (path loss, *PER*, transmission delay) with abstract parameters for the conditions emulation. E.g., a path loss model (ref. Subsection 2.2.1) is used to calculate the received power at a certain distance, including empirically-determined in-house wall attenuation values and a shadowing component (cp. (1) in [113]). The *PER* is used to introduce statistical probability for a successful frame transmission. Thus, the channel parameters are applied to the transmitted frames on the wired testbed network to decide whether to eventually deliver or discard received frames.

Elsner et al. [114] propose a *model-based* approach for emulating the node signal processing in software using an SDR environment. This enables switching between real software execution in a pure virtual software radio network and on the target *SDR* hardware. They model the radio front end as well as the radio channel with software components. A *channel matrix* component operates synchronously to the *SDR* nodes and simulates signal propagation effects (e.g., Doppler shift or fading) for each individual channel between a pair of nodes. The effects of the propagation path are modeled individually by a time-variant impulse response. Thus, the channel matrix becomes increasingly complex for large-scale virtual networks.

The following Figure 3.3 compares the selected approaches in an overall view based on the properties introduced in Subsection 3.2.1. Note that the scale does not represent defined quantities, but a linear level of performance from very poor (center of the radar plot) to excellent, compared to the highest property values of the selected approaches[1].

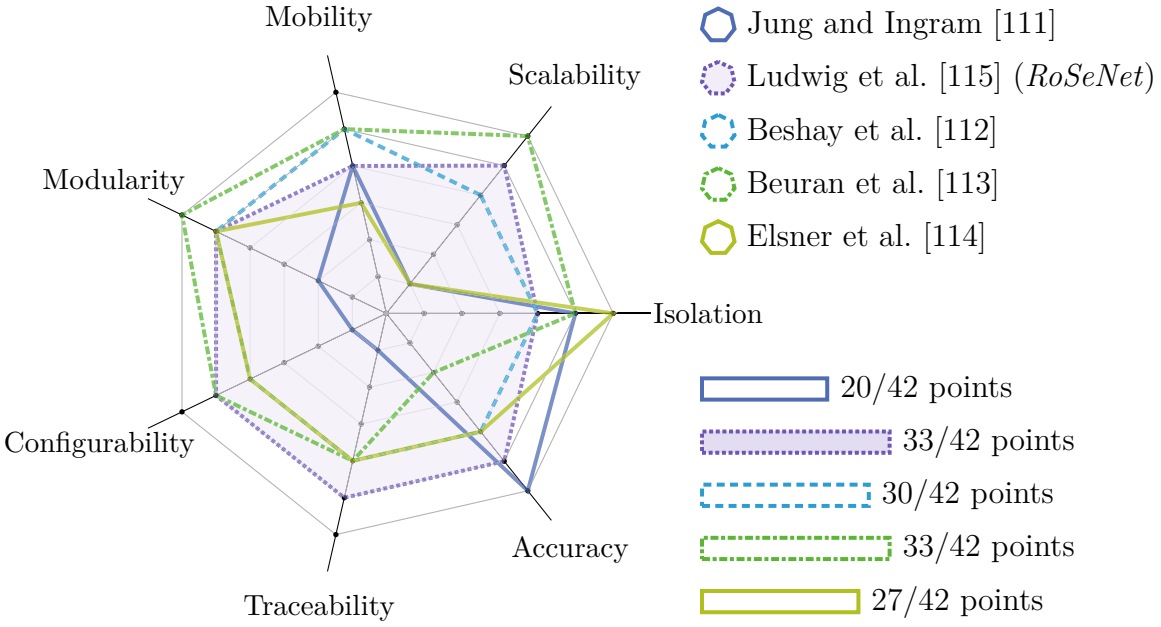


Figure 3.3: Analysis of radio link emulation systems and approaches based on the properties performance level. The overall performance is additionally represented by a score and highlighted for the *RoSeNet* emulation testbed[1].

To briefly summarize the current approaches, it can be stated that radio link emulation is a very heterogeneous field of research with partly very different approaches. Considering the overall performance on the basis of this requirements there are three systems that are in the particular focus, namely *RoSeNet*, *WiNeTesTer*, and the emulation testbed introduced by *Beuran et al.* These systems are particularly characterized by their broad applicability, because they are not oriented to specific communication technologies or concrete issues. It can also be recognized that the most accurate testbed-like approaches tend not to be very scalable. An exception is *RoSeNet*, which is analyzed in more detail subsequently, since it is the experimental platform of this thesis. Except for the model-based systems in [114] and [118], all approaches rely on several real hardware resources (either emulation hardware or testbed infrastructure) which makes them particularly costly, but very accurate methods of network evaluation.

[1] The exact interpretation of the radar plot evaluation is given in Table A.2 in Appendix A.

RoSeNet Emulation Testbed

RoSeNet [115] was emerged from a joint project of Fraunhofer IIS / EAS and dresden elektronik company. The elementary goal of *RoSeNet* was to provide procedures and methods for the design of robust, reliable, and functionally safe wireless network solutions in the context of *Low Rate s (LR-WPANs)*. It features a low-power wireless technologies hardware test platform (hereinafter referred to as *RoSeNet*) developed by dresden elektronik for the analysis of the robustness in load scenarios for large-scale *WSNs* with a size of up to 1000 radio modules.

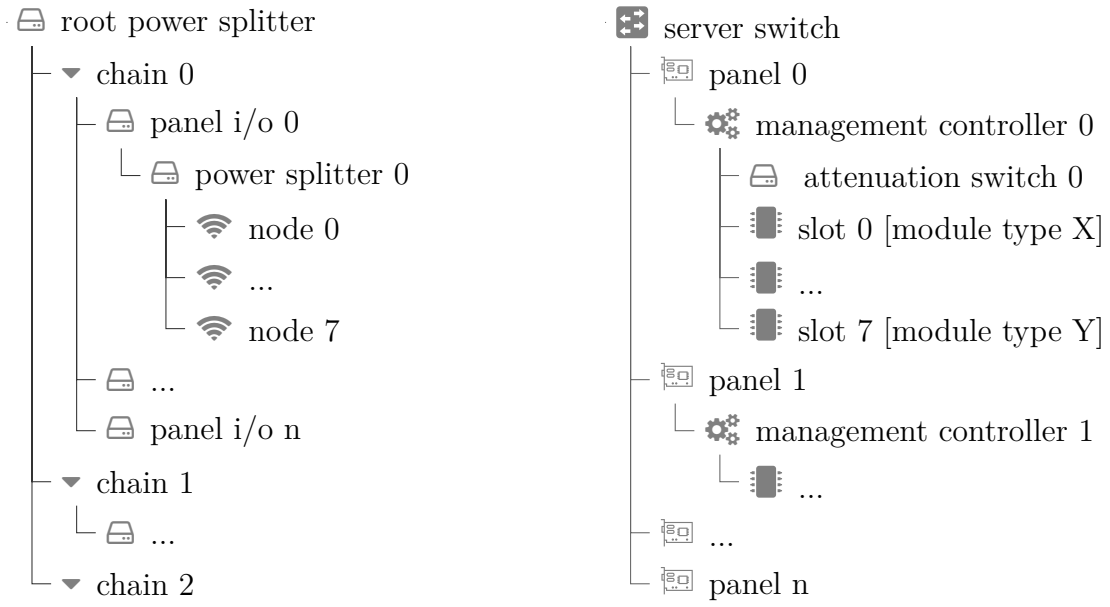
Figure 3.4: *RoSeNet* radio emulation and test platform hardware panel (taken from [55])

Figure 3.4 shows a photo of a single *Emulation Panel* of the entire *RoSeNet* platform. All "wireless" nodes on the hardware system are interconnected through a controllable coaxial cable radio environment and do not use actual external antennas. As a representative of a *class A* emulation platform, all other components, such as sensor node hardware, communication interfaces, network stacks, and application firmware, are real.

Testbed Hardware and Control Architecture

RoSeNet has a very modular hardware architecture that is composed of individual hierarchy levels, but it is not necessarily bound to a fixed structure. A single *Emulation Panel* forms the basic component of the emulation hardware system. It is responsible for the inclusion and control for up to eight radio modules. A panel can either be operated stand-alone or in association with others. Multiple panels are interconnected in two different hierarchical networks: a *RF* and a Control Network. The basic characteristics of these network views on the hardware architecture and its components are analyzed in Figure 3.5 and explained below.

The basic RF *Network* hierarchy is presented in Figure 3.5a in which coaxial cables connect all components. From the *RF* point of view, a panel is a compound of eight nodes whose wireless interfaces are connected to one another by a *power splitter* (power divider). The resulting signal path is linked to the panel output and can be adjusted by digital step attenuation. There are *BNC* connectors on both interfaces of the panel *Input/Output (IO)* that connects to adjacent panels. A series connection of multiple panels forms a *RF chain*. Eventually, the entire *RF* network is formed by multiple chains that are attached to a *root power splitter*.



(a) *RoSeNet RF* network (b) *RoSeNet* control network device hierarchy

Figure 3.5: *RoSeNet* hardware architecture

In this overall hierarchical configuration, the maximum number of "wireless" interconnected nodes in a *WSN* test scenario can scale up to 1000. This is because with each connected panel in a *chain*, the signal attenuation to a distant node increases. Due to common hardware values for the transmission power and receiver sensitivity of common radio modules, the radio signal can no longer be detected at some point. The node network hierarchy can also be used partly or even freely configured with power splitters and coaxial cables in other topologies.

From the control servers point of view, there is a *Control Network* (see Figure 3.5b) based on a switched Ethernet *LAN* which connects to all emulation panels. Hence, every panel can be attached via Ethernet-Switches by its *MAC* address. A management server is responsible for controlling each individual panel and coordinating the panel access within a test sequence. For the communication with each panel, a simple panel command protocol over the common *TCP/IP*-Stack is used. A single panel features a variable operating voltage for the module slots and a power consumption measuring circuit. For each module slot, certain node interfaces can be controlled, e.g., the *Universal Asynchronous Receiver and Transmitter (UART)* and analog/digital *IO*. In addition, radio modules on the panel slots can be of different device types because they are connected via individual interchangeable adapter circuit boards. There are control commands specified for the panel and each single node on the module slot.

3.4 | Limitations of Radio Link Emulation

This section concludes the analysis of emulation for resource-constraint wireless communication networks by showing the limitations of pure network emulation. As, for example, the criteria-based tool comparison and overview with advantages and disadvantages in [38, p. 88] is too general, the analysis in this section provides a specific overview of the limitations with a particular focus on radio link emulation with respect to current research questions. Some general limitations, presented below, are complemented by specific constraints in the corresponding subsections.

As mentioned before, the most limiting factor is probably the costliness of the system hardware to enable accurate hardware-based channel emulation. Even with virtual approaches based on a testbed infrastructure, the enormous demand for resources, especially for large *WSN* and *IoT* deployments, drives up costs. Since commercial channel emulators support fine-grained emulation of the wireless channel only among a very small number of nodes, evaluating new device, implementations or applications are limited in network scale. Moreover, considering the hardware-based emulation, it is a very difficult task to provide internal isolation in a testbed-like system, e.g., to avoid multipath effects of radio signals over the wired control network.

As in digital and software-defined network emulation approaches, e.g., [112] or [114], the *RF* path is digitized, signals could be manipulated with full control. The high computational demands, however, results in low scalability, very complex and still abstracted algorithms for the incorporation of propagation effects or interference, and coexistence with other technologies. Furthermore, these concepts focus on network condition emulation and require real transceiver hardware and software, which at the same time excludes aspects of a flexible protocol stack interaction.

3.4.1 | Cross-Layer and Cognitive Radio

Although radio interface emulation is based on real protocol implementations and even radio channel emulation relies on real transceiver chips and interface hardware, they lack in flexibility of collaborating with configurable radio interfaces and higher protocol layer features. Considering the approaches in [113] and [114] or the generally limited scope of *SDR* concepts (cp. [110, pp. 40f]), only the *PHY* and the radio channel should be considered for accurate modeling. Furthermore, the link layer is often based on an implementation of the standard as a *NIC*, whereby changes on the *PHY* parameters are not possible by design. As highlighted in this thesis' introduction chapter, there is an increasing demand for *cross-layer* evaluation approaches allowing information exchange between all higher communication layers with the physical communication interface. Several problems created by wireless links are addressed in [49] which cannot be evaluated

with pure radio link emulation due to the lack of interactivity and configurability of the higher layers or *NIC*-based implementations (cp. *MAC* prototyping survey in [123])

On the other hand, radio link emulation provides a very good basis for cognitive radio networks ([124]). As mentioned before, without incorporating the effected higher layer protocols and applications, there is a huge tradeoff in terms of flexible experimentation. Therefore, from our point of view, there is a great need of expansion and coupling radio link emulation approaches with flexible higher-layer modeling.

3.4.2 | Real-Time Scalability and Mobility

When demanding high reality with real-time execution, the best results can be achieved in testbed-like channel emulation systems of *class A*, but they are often not very scalable (cp. [111], [119], [116], or [120]). *Class B* emulations, on the other hand, focus on emulating the communication interfaces for resource constraints of wireless devices which can become difficult to emulate on general purpose single host systems like in [114]. In distributed approaches at least one central controller is needed to synchronize the infrastructure components, feed them with commands, and process the result data.

When using model-based calculations for the environment emulation, real-time execution is limited depending of the level of abstraction. Especially software-based channel emulation reaches the limits of real time computability with a large number of nodes at the inputs and outputs. In large-scale networks there must be a high degree of distributed control, parallelism, and modularity which is complex to achieve in channel emulation. The problem becomes even larger if node mobility is to be taken into account because then channel parameters must constantly be recalculated with complex algorithms at runtime. Mobility calculation in real-time emulation is practically unfeasible in analog hardware setups because it requires highly complex calculations for larger topologies.

4 | Parallel Simulation and Emulation

It inevitably follows from the findings of the previous chapters that certain approaches of ongoing research issues cannot be operably evaluated by pure simulation or emulation. Hybrid evaluation techniques can achieve great added value for certain evaluation questions and thus mitigate the highlighted practical as well as methodological limitations of stand-alone techniques. There are a number of single-purpose evaluation tools which provide a specialized solution for a specific problem. As a result of our analysis, we propose a novel general-purpose conceptual approach in this chapter that combines protocol simulation and hardware-based radio link emulation, called *Split-Protocol-Stack Wireless Network Emulation* with *Radio-in-the-Loop (RIL)*.

Subsequently, we present the terminology of the relevant state-of-the-art coupling techniques and our derived *RIL* methodology with regard to wireless communication systems, sensor networks, and the *IoT*. We highlight fundamental requirements, current research questions, and problems addressed by *RIL* which seamlessly leads into the conceptual design and architecture of our *Split-Protocol-Stack* strategy. We introduce a generic splitting into different evaluation levels and communication system domains. Based on three main challenges, we show respectively how the identified research questions and problems are addressed in this thesis. Furthermore, the singularity of our strategy is highlighted and contrasted by a specific survey of the most significant related approaches to hybrid network evaluation, which deal with comparable research challenges. Finally, we discuss the main benefits, use-cases, and scenarios when using parallel simulation and emulation based on our example and prototype architecture.

4.1 | Methodology and Requirements

With consideration of hybrid simulation systems that integrate real or emulated hardware resources, established key concepts, such as *Hardware-in-the-Loop (HIL)*, are influencing the field of network evaluation, but they are not applied for their original purpose. These techniques must be adapted and enhanced for their use in wireless network evaluation. While the basic concepts of simulation and emulation have already been presented in Chapter 2 and 3, some terminology, methodical delimitation, and classification of parallel simulation and emulation methodology have to be pointed out in the following. Moreover, we introduce the *RIL* terminology for the *Split-Protocol-Stack* approach and analyze the most important problems and requirements of coupling simulation systems with real hardware.

4.1.1 | Terminology and Methodical Delimitation

The concept of *HIL* simulation has its origins in the field of testing embedded systems, especially in the automotive industry, where prototypes are fed with simulated data generated from a virtual environment. Hence, the *HIL* simulator replicates the operating environment of the real electronic component and processes the results generated by the hardware for further analysis. In the context of network modeling and simulation this concept is applied rarely, which means there is no standard procedure for implementing *HIL*-based network evaluation. In general, *HIL* network simulations incorporate physical network hardware as integral parts of a larger simulation system. Practical use cases can be found in interoperability or performance tests of varying vendor hardware platforms [39, p. 114], for example.

The terminology of the *cluster simulation* is very similar to the *HIL* simulation approach. In the area of network evaluation, e.g, the simulation environment is connected to the system-on-chip hardware, a node microcontroller, or a network of nodes via a communication interface (cp. [125] for an example use case). The difference to *HIL* is that the simulator is only responsible for generating data packets as input signals to the system under test. Since no return channel exists to feed data from the hardware into the simulation, it is only possible to verify the behavior of the system under test. Thus, it is not possible to increase the simulation accuracy. Both *HIL* and cluster simulation fall under the category of real-time simulation and usually require high-performance hardware and real-time operating systems.

In a *co-simulation*, several simulation programs are used which feed each other the data for the next simulation step. Co-simulation is mainly used when a simulation model is needed for the analysis of special effects whose modeling requirements go beyond the functional scope of a single tool. With respect to wireless network evaluation, co-simulation in practice is mainly applied to extend network simulations with other networking aspects, e.g., mobility in *Vehicular Ad Hoc Networks (VANETs)*, cp. [126]. Furthermore, decoupling complex calculations of physical effects from the network simulation can help to increase the scalability of the simulation.

The terminology of a *RIL* simulation was first introduced by our practical investigations of integrating real radio transceiver hardware into model-based simulation software in [55]. The parallel simulation and hardware-based emulation is a combination of the *HIL* and co-simulation methodology. The sensor network to be analyzed, i.e., the deployment environment of the hardware used, is represented by models in the network simulation and fed by the results of the computations on the real hardware. Thus, the accuracy of the real hardware improves the overall simulation credibility with this method. To roll out a basis for common understanding of our classification in the area of wireless networking we define some terminology for parallel simulation and *HIL* based emulation.

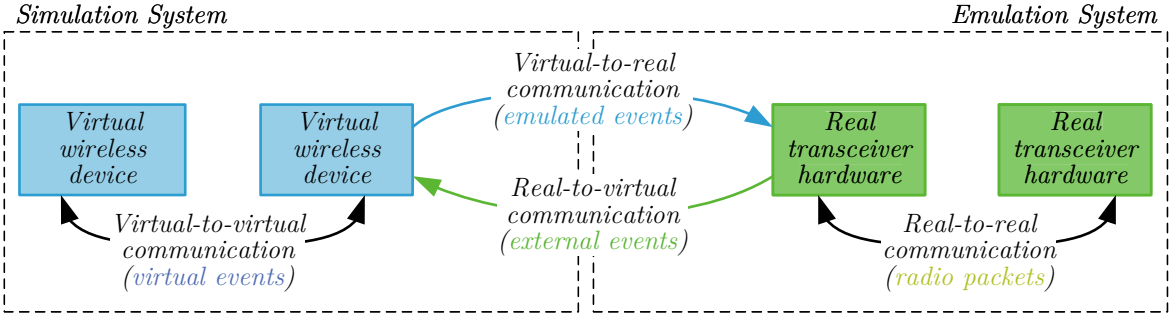


Figure 4.1: Definition of terms and communication relations for the parallel simulation and emulation *RIL* methodology.

Figure 4.1 introduces the relevant terminology for the parallel simulation and emulation setup. There are different types of communication whose messages are called *radio packets*, *virtual events*, *emulated events*, and *external events*. While *virtual* events correspond exactly to those in the pure simulation and *radio packets* represents real data transmission between real radio transceivers, *emulated* events refer to those events that are generated virtually and processed by the emulation system on the real transceiver hardware. On the other hand, *external* events are generated from real hardware resources and transmitted to the simulation system.

4.1.2 | Coupling Problems and Requirements

The requirements for using simulation and emulation in parallel are fundamentally the same as introduced for emulation systems in Section 3.2. Real-time capability is a key requirement for *HIL* simulation scenarios, as the hardware resources are running in a continuous environment. Thus, *HIL* is delay-sensitive by design, which means an experiment setup is often bound to a fixed transmission rate in network traffic with low latency and minimal jitter. This requirement becomes hard to achieve when increasing the complexity of the models on the simulation part, as already discussed with the limitations of simulation environments in Section 2.4. In addition, there are some further requirements or research problems which occur when incorporating substantially different evaluation methods to a hybrid approach. These problems are summarized and discussed below, derived from various related work *HIL* modeling aspects, e.g., in [127], [66], [128], and Section 4.3.

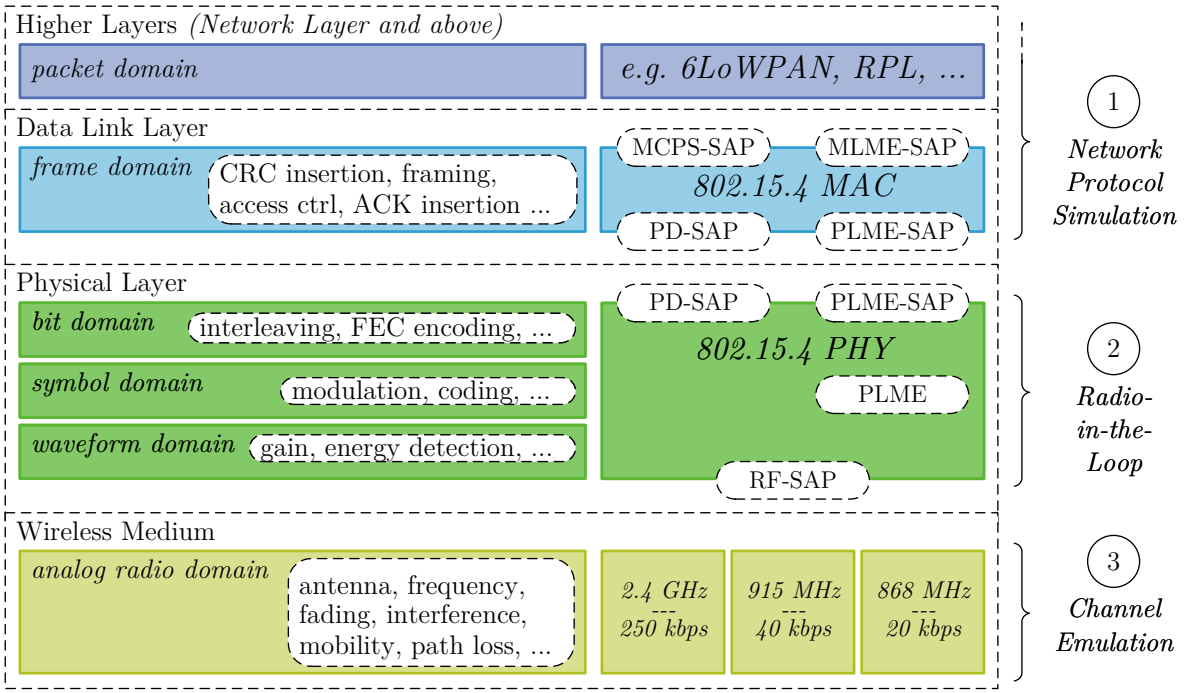
- **Interfacing** between the simulation and the hardware system is substantially for all *HIL* approaches, as it enables the message exchange and control. For a widely applicable and general solution, the interface must support different simulation environments and allow the exchange of the above mentioned *emulated* and *external* events between real and simulated system parts. A generic interface solution is able to support different physical interfaces based on standardized and well-known

data formats. Thus, both standard-conform wireless transceiver hardware and virtual or model-based emulation can be attached to the simulation. Furthermore, *interface abstraction* can enable a configurable or exchangeable interfacing, e.g., for accelerating communication based on a customizable level of detail or message priorities.

- **Synchronization** is a key issue when coupling a simulation system with another system to run in parallel. The synchronization should prevent that the events from the two systems arrive too late for the execution. Two different and commonly used event orderings are the *receive order* and the *timestamp order* [129]. They must ensure the causal order of the events caused by various logical processes. When running a real-time *HIL* simulation, the correct order of scheduled events is maintained by the simulation scheduler. Since data transmission and computation latency though produce different times on different systems, a precise and accurate synchronization of all logical processes in the whole evaluation setup is needed additionally. Furthermore, an evaluation system with several *CPU*s on different hardware systems (usually within hardware-based parallel simulations and emulations) runs with independent physical clocks, each at slightly different rates.
- **Scenario modeling** as well as scenario generation, is a central task of the simulation software, according to the common definitions of *HIL*, cluster, or co-simulation. While network simulators provide a dedicated toolset for configuring a simulation (e.g., the configuration files for the simulation kernel and model parametrization in *OMNeT++*/INET), a hybrid simulation approach relies on further configuration management of the coupled system, e.g., the radio link emulation. Usually, simulation models and systems are parameterized by configuration files to facilitate parameter studies and to enable reproducible research. Otherwise, default or preconfigured parameters are applied to the simulation setup. The automatic *generation* of a scenario, e.g., for a variable number of nodes in a specific topology, is another key task of the network simulator (cp. topology generation in Section 2.2).
- **Hardware abstraction** of specific nodes increases the modularity and configurability of the evaluation system. This allows device hardware from different manufacturers to be set up with the same software and interfaces. Also in embedded wireless devices, these functions are performed by operating systems. Furthermore, the testbed hardware infrastructure also needs abstraction to decouple universal system features from specific hardware components.
- **Resource allocation** of nodes and infrastructure hardware is closely tied to the scenario configuration and the infrastructure-hardware abstraction. It enables a precise mapping between a certain simulation scenario and the corresponding node hardware. Since the device hardware and network configurability in hardware-based radio link emulation has fixed boundaries, this is a problem of high complexity with limitations regarding accuracy and feasibility.

4.2 | Split-Protocol-Stack Wireless Network Emulation

The *Split-Protocol-Stack* approach is a result of our analysis, practical experimentation, and contributions ([54–56, 58, 130]) and constitutes the heart of this thesis. This strategy and architecture for simulation-driven wireless network evaluation is superordinate to the successively following research challenges and thus forms the methodological framework of all further details, individual contributions, and discussions in the subsequent chapters. Based on aforementioned naming conventions introduced with the parallel simulation and emulation methodology in Section 4.1, it is called *Split-Protocol-Stack Parallel Simulation and Emulation with RIL*.



(a) Wireless Transmission System Domains (b) Reference Protocol Stack (c) Evaluation

Figure 4.2: The Split-Protocol-Stack architecture with the protocol stack decomposition **(a)**, the reference protocol *IEEE 802.15.4* **(b)**, and the evaluation levels **(c)**.

Observing the protocol stacks of wireless transmission systems, a division of the network interface into the Physical Layer and the Data Link Layer (especially *MAC* functions) is apparent, regardless of the specific wireless technology (cp. clearly assigned responsibilities explained in Section 2.2). The *Split-Protocol-Stack* strategy applies exactly this formal approach of different responsibilities as its concept and assigns these tasks to different evaluation levels, as illustrated in Figure 4.2. In addition, a comparison with the *IEEE 802.15.4* example is given which we use as reference protocol stack. The *DLL*, mainly covered by the *MAC* features. All further (hardware-independent) higher layer protocols are assigned to the pure virtual protocol simulation (level ① in Fig. 4.2). We involve

the *PHY* layer specifications and the main radio transceiver features for transmitting standard-conform radio packets over real radio interfaces (level ②). As this evaluation layer covers most of the radio transceiver hardware, it represents the *RIL* methodology. The *Split-Protocol-Stack* builds upon a hardware analog radio channel emulation for resource-constraint wireless communication systems (level ③).

We further decompose the protocol stack layers into sub-domains (ref. Figure 4.2a) to indicate the wireless lower layer functions, presented in Section 2.2. The *frame* and *packet* domains of wireless nodes can be represented in this framework with sufficient accuracy when evaluated in discrete event simulation. As most simulation models and frameworks consider events at the frame level, we split the evaluation at the layer boundary to the *bit* domain. Thus, the bit-level and below operations are processed at the *RIL* transceiver hardware. The *symbol* and the *waveform* domain define the steps to transform a bit or several bits into a modulated symbol and transmit it via digital-to-analog conversion on the radio interface. The *analog radio* domain represents the wireless transmissions of the nodes in which radio propagation effects and characteristics of the wireless medium are emulated.

With respect to the realization of the *Split-Protocol-Stack* approach and architecture, we give an overview about three fundamental research challenges associated with the three different evaluation levels ①, ②, and ③ (ref. Figure 4.2) in the following. Respectively, the addressed coupling problems and requirements of the *RIL* methodology, introduced in Section 4.1, are highlighted. Full details about our specific contributions associated with these research challenges as well as the evaluation results are presented in the subsequent Chapters 5, 6, and 7.

4.2.1 | Pseudo-Real-Time Network Simulation

The *pseudo-real-time* network simulation is our suggestion to enable scalable *HIL* simulation coupling needed for the *Split-Protocol-Stack* approach. As discussed in Section 2.4, real-time capability can become a complex problem for accurate and scalable model-based network simulations. Because of this we introduce enhancements of the simulation scenario modeling and an extended real-time simulation scheme that enables the connection to real-time hardware.

Scenario Modeling and Configuration

The overall scenario modeling and configuration for the *RIL* wireless network simulation, running virtual higher layer protocols (ref. frame and packet domains, level ① in Figure 4.2) on physical real node hardware is based on the simulation environment. In order to enable standard-conform access over the service access points and exchange the

necessary service data units with the real hardware transceivers, an accurate representation of a desired link layer protocol specification is inevitable. Therefore, we rely on our high accuracy *IEEE 802.15.4* simulation model [96, 97] which is modeled in *OMNeT++*, strictly according to the specifications and general modeling guidelines of *IEEE 802.15.4* [9]. For the *RIL* concept, we adapt and add emulation extensions to the simulation model for the scenario, dispatching, and scheduling introduced in [54]. All details of the protocol and scenario modeling, configuration, and execution in the simulation environment are presented in Chapter 5.

As modeling, the scenario configuration and the execution control are also handled by the simulator. Due to our parallel simulation and emulation environment, the emulation domain also needs parameterization depending on the simulation scenario. Since congruently manual configuration of the emulation system can lead to a challenging and cumbersome task, we introduce an assisted emulation scenario configuration. We implemented prototypical parsers and generators for the emulation control system which accepts the simulation configuration files (esp. the *OMNeT++* `.ned` network definitions and `.ini` configurations) as inputs and generates the appropriate emulation system setup for a given scenario. Important parameters that must be configured consistently for the simulation scenario by the emulation system are, for example, the specified (geographic) position, the device type of a node, the *PHY* technology, and the radio channel parameters (e.g., frequency band, channel, and modulation scheme).

Synchronization Concept

Due to the splitting between the *MAC* and the *PHY*, which are usually implemented to keep latencies in function calls between the two sub-layers as short as possible, the time-synchronization between two different domains and systems becomes a fundamental problem and bottleneck. Since the emulation hardware is connected via *LAN* requires communication with other control computers, the required microsecond accuracy cannot be met, even over a dedicated network connection. However, if a constant, even (within limits) arbitrary, delay between the components is achieved, this can be recalculated afterwards, according to our synchronization concept.

We introduce a pseudo-real-time scheduling as solution approach and basis for the synchronization called *Real-Time-Shift. Pseudo* means that the simulation runs at real-time, but the simulated real-time is temporarily manipulated by waiting times during the transmission of emulated and external events. Accordingly, the (virtual) simulation time is permanently *shifted* back and forth due to the event transmission delays. However, the basic requirement is a precise clock synchronization of all attached components in a dedicated *LAN*, e.g., using the *Precision Time Protocol (PTP)*.

4.2.2 | Radio-in-the-Loop Wireless Transmissions

With *RIL* wireless transmissions, we assume that real radio hardware (components) represent the radio transceiver interface of a communication system that is able to send (*TX*) and receive (*RX*) real radio packets. On the other hand, the hardware must be able to process several *PHY* functions, e.g., carrier sensing or *ED* (cp. *PHY* modeling responsibilities in Section 2.2). Thus, the use of real transceiver hardware in wireless network simulations allows for an accurate representation of the *PHY* domains (e.g., symbol and waveform domains, level ② in Figure 4.2). Furthermore, the radio hardware takes over the interface function from the simulation domain to the real radio environment in the emulation domain.

Interfacing the Radio Channel

The type of coupling can be accomplished either using a *gateway* architecture or a *one-to-one mapping* of bridged simulated and real nodes. While the former is primarily suitable to extend a real network by simulated nodes, the latter can achieve the increase in simulation accuracy aimed at in this work. From the simulators point of view, the emulation domain is considered as a *black box* and, on the other hand, the *RIL* gateway nodes only know the interface which generates the message input data. For each bridge node, a physical data communication interface to the control subsystem is necessary.

We have chosen the *PCAP* container as transmission control and data exchange format among all involved subsystems, not least because it supports the individual extensibility of frame formats with embedded optional fields and the capability of carrying data frames from multiple network interfaces within one single data stream. This coupling methodology makes the *Split-Protocol-Stack* a universally valid strategy for experimenting with different radio hardware, simulation frameworks or protocol models, and hardware-based radio emulation setups.

Abstracting from Node Hardware

In order to transmit simulated protocol data over the radio channel and vice versa an accurate execution of the *PHY* functions is needed in the *RIL* hardware. These functions depend on the concrete link layer protocol standard properties. As various vendor devices are suitable for this which are slightly different in core in terms of performance, features, and energy consumption, we implemented an independent *stream processing* and *interface control* which is capable for the transparent handling of multiple interfaces with virtual, emulated or real hardware devices. With this abstraction, we present and experiment with two different approaches of modeling the *PHY* layer, both related but not limited to the *IEEE 802.15.4* link layer protocol specification. All conceptual and practical details of *interfacing* and *abstraction* are explained later in Chapter 6.

- **Real-Time Operating Systems** for wireless devices (e.g., *TinyOS*, *Contiki*, or *RIOT*, introduced in Section 2.3) support different vendor device hardware. We developed a bridge transceiver software in *Contiki* and *RIOT* which handles incoming messages over a serial line (*UART*) for executing radio driver processes and eventually transmits real radio packets.
- **Real-Time Streaming with SDR** is a more flexible solution for experimenting with *PHY* operations and parameters, since available vendor device hardware supports only a reduced feature set of a given link layer specification. With our *SDR* transceiver model in *GNU Radio* we enable *PHY* modeling diversity with respect to a single or even multiple radio technologies. Thus, we are able to physically switch channels, modulation schemes, or sensing parameters.

4.2.3 | Radio-in-the-Loop Analog Radio Channel Emulation

The real packet data from the *PHY* are transmitted as analog signals in the controllable *RF* channel emulation environment and can thus be manipulated in real-time. Network emulation testbeds, such as the *RoSeNet* hardware subsystem, are particularly suitable as closed systems. Besides the control and data communication to the hardware nodes, the main tasks of the emulation hardware are on the one hand to provide resources for a given scenario, and on the other hand to emulate the effects in the analog radio domain (level ③ in Figure 4.2). The most important feature of such systems is the configurable long-term fading.

Radio Topology and Mobility Modeling

Based on the attenuation values, a static network is obtained which becomes a network with mobile nodes when changing these values permanently. For pseudo-random mobility, fixed intervals can be defined for each attenuator and continuously changed during the system runtime. We discuss several suggestions for extending the emulation capabilities of network emulation testbeds, in particular the *RoSeNet* hardware, to increase the mobility support which need particular and invasive adjustments of the emulation hardware.

Resource Allocation

The most hardware-based channel emulation approaches provide configurable *RF* components (e.g., step attenuators) between the nodes, but they do not discuss how to configure the radio frequency hardware equipment for a given scenario (e.g., a multi-hop network) with relative physical positions and distances of the nodes. With our work, we provide first results of an approach for solving this complex task based on a *MILP* optimization scheme in Chapter 7.

4.3 | Analysis of HIL and Co-Simulation Approaches

Well-established simulation tools, e.g., *OMNeT++*/INET, *ns-2/ns-3* or J-Sim [131], enable real applications within the network simulation by implementing an *API* to the transport layer protocols (*TCP* and *UDP*) or a raw socket interface to capture and transmit packets from and to the simulators host network interfaces (e.g., [66]). These embedded emulation features of simulators often form the prerequisite for parallel-coupled *HIL* simulations. Nevertheless, there are no general modeling guidelines for *HIL* and co-simulation/emulation systems. It is very difficult to define an appropriate classification, since most approaches develop a very specialized system depending on the research question pursued. In the last two decades several diverse methods have been proposed which are reviewed in more detail next. We compare the specific contributions with the *Split-Protocol-Stack* approach and give a posssible classification of the approaches.

Co-simulation approaches that do not focus on enhancing the protocol stack accuracy but only virtual-to-virtual communication, e.g., coupled network simulation and road traffic simulation with *Veins* in [126], are not considered in the following analysis. In [132] has been shown, how to enable application-specific interfacing between automotive and network simulation models, e.g., for exchanging traffic positions, scheduled within synchronous message cycles, but this is in contrast to a protocol stack based *HIL* concept. There are a few *HIL* approaches that focus only on code generation for specific node hardware platforms (cp. [133] or [134] for example) or for the verification of specific protocol functions for single devices under test (e.g., [135]). They are not suitable for scalable network evaluation. Concepts of *SDR* Hardware-in-the-Loop-based channel emulation primarily focus on software radio testing, completely isolated from the communication protocol stack of wireless systems and application-related processes, e.g., featuring a simulation of the point-to-point performance with pure virtual *RF* front-ends [114], or a *FPGA*-based digital wireless channel emulation [120].

When looking into practical contributions in the field of protocol-stack-related network evaluation, a distinction of *HIL* and co-simulation approaches is emerging based on how hardware resources are represented in the overall setup. These can be either fully *simulated* in a co-simulation system design, integrated by real, or virtual components of a general purpose *host* computer system or correspond to real *external* devices. A further distinction can be made according to how protocol implementations of network devices are represented and what role simulation plays in the overall setup. This can be real hardware protocol implementations used in a *simulated network environment* or *simulated protocols* that interact with the network interface hardware. Related approaches can be assigned to this distinction and have many similarities in their characteristics. Therefore, we make a division into two basic strategies in the following, each with the three different ways to incorporate hardware resources.

4.3.1 | Real Protocol Implementations in Simulated Networks

simulated hardware	*virtualized hardware*	*external hardware*
Jung et al. [136]	Kato et al. [137]	Staub et al. [138]
Riliskis et al. [139]	*Weingärtner et al.* [140]	*Unterschütz et al.* [141]
Zhang et al. [142]		*Wehner et al.* [143]

Table 4.1: Overview of HIL and co-simulation approaches running real or emulated protocol implementations in simulated networks

Running real-world protocol implementations in simulated networks has been the most widely considered approach with the most practical research over the last two decades. They allow the execution of unmodified wireless communication software running on real hosts in a virtual protocol stack network simulation, but without enhancing the modeling of the *PHY* or the radio environment, which is one of the most limiting constraints (cp. Section 2.4). The approaches in [136], [139], and [142] demonstrate the emulation of code execution on hardware models and the use of the peripheral hardware and instruction-set processor emulation coupled with discrete event simulation to provide accurate measurements of the nodes hardware dependencies, e.g., the energy consumption on the transceiver hardware. Nevertheless, these systems are specific to node hardware platforms and/or operating systems (e.g., *AVR* hardware in [136] or *TinyOS* in [142]).

According to the *NSC* approach in [38, p. 88], real software can be directly integrated in network simulators on host systems. This is possible under certain conditions by modifying the existing software implementations and compiling them as a shared library. Using (standard) host computer network stacks in simulations enables the execution of unmodified real host network stack software [137, 140] combined with virtual node data transmission within the simulator. As the scheduling of Weingärtner et al. [140] is based on sequentially executed equal *time slices* for the participating devices in *ns-2/ns-3*, it cannot be applied to a true parallel real-time radio channel access of multiple devices.

In [141], Unterschütz et al. present a hybrid testbed based on connecting testbed nodes to an *OMNeT++* simulation. The application layer implementations can run on real nodes in which the below layers are simulated in *OMNeT++*. A similar approach is taken by the work of Staub et al. in [138]. Here the upper protocol layers (*NWK* and above) run on real nodes with virtual interfaces connected to a *virtual mesh*. Wehner & Goehringer [143] include real IoT radio interface hardware that exchanges real radio packets with the simulation, but primarily focus on the translation of proprietary protocols and application control messages (for *Z-Wave* and *EnOcean*) to connect different vendor devices. There are no real-time dependent timing considerations for scheduling events in scalable network scenarios and no accurate modeling of the radio channel.

The approaches listed in Table 4.1 are analyzed in overview to compare the properties,

which can be seen at a glance in Figure 4.3. The selection of approaches compared is based on methodological diversity - the most divers ones are contrasted[1]. What can be figured out in is that the overall system performance of all approaches is very similar, besides the very specific setup in [143]. No general methodology for a *HIL* evaluation can be derived from this approach by Wehner et al.

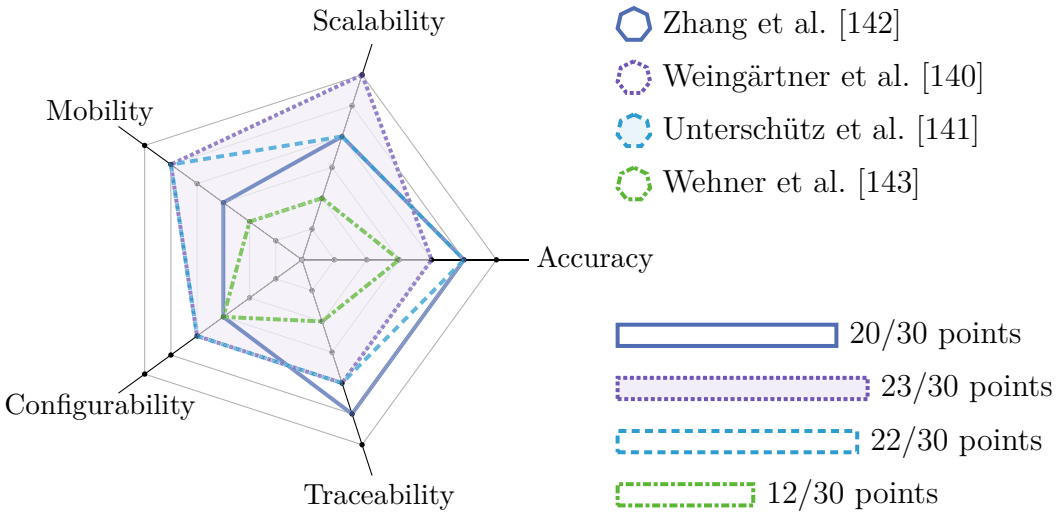


Figure 4.3: Overview analysis of approaches running real or emulated protocol implementations on hardware resources in simulated networks based on the properties performance level, additionally represented by an overall score[1].

When considering the traceability and accuracy of the entire modeling domains, the approach of Zhang et al. [142] turns out to be very detailed and accurate, but it is difficult to parametrize the models, configure a scenario, or analyze protocol-related features at a high level because they are implemented in the node firmware. These are certainly not ideal conditions for research in cognitive radio and cross-layer systems but for performance evaluation of *TinyOS* network applications. The characteristics of Staub et al. [138] and Weingärtner et al. [140] do not differ significantly, as they implement very similar approaches for *OMNeT++*/INET and *ns-2/ns-3*, which are both suitable for wireless network simulation (cp. Section 2.3). The overall configurability in [140] is high because the radio link can flexibly be parameterized in model-based simulations. In contrast, however, the adaptation of the host network protocol stacks is only possible to a limited extent. As the real-time scalability is very limited due to complex radio link simulation models, with *SliceTime* [144] Weingärtner et al. mitigate this issue by relieving the network simulation from its real-time constraint. They demonstrate a scenario with 15000 simulated nodes which executes about 15 times slower than real-time due to the high event load. A big advantage of the approach of Unterschütz et al. [141] is the use

[1] The exact interpretation of the radar plot evaluation is given in Table A.3 in Appendix A

of the *CometOS* operating system both in the simulation and in the testbed domain. Since the hybrid sensor nodes in the testbed communicate over the same protocol stack, a high overall accuracy can be achieved. The scalability of setups with hybrid nodes though is not entirely clear and the reproducibility of evaluation scenarios is limited to the simulated part. The latter can be extended when running the nodes on a network emulation testbed like *RoSeNet* (cp. Section 3.3).

4.3.2 | Simulated Protocols on Real or Emulated Networks

simulated hardware	*virtualized hardware*	*external hardware*
Mittag et al. [52]	*Obermaier et al.* [145]	*Ding et al.* [124] Klingler et al. [146] *Split-Protocol-Stack*

Table 4.2: Overview of HIL and co-simulation approaches running simulated protocols on real or emulated networks

There are only a few approaches that focus on the simulation of communication protocol designs and application software which simultaneously emulate the impact of hardware interfaces and radio transmissions at cycle-level accuracy. Mittag et al. in [52] present an appraoch to bridge the gap between network protocol and *PHY* signal simulation. They show how to integrate a simulator software for *OFDM*-modulated *IEEE* 802.11 communications as the *PHY* and radio channel into the *ns-2/ns-3* network simulator. As the *PHY* simulator is implemented as a state machine according to the *IEEE* 802.11 specification, it transforms the data input bits into the time domain samples, where several propagation models can be used for the radio transmissions. They validated their experiments against the wireless network emulator [116] presented in Section 3.3.

Enabling real transmissions from the simulator computer system by using *SDR* is applied by Obermaier et al. in [145] who present a single node device testbed for *VANET HIL* simulation based on *IEEE* 802.11p. They split the network architecture vertically to implement a virtual network bridge for transmitting only the *PHY* data frames via a single gateway node, called *physical twin*, within an uncontrolled radio environment to the real device under test.

Running external radio transceivers connected to the network protocol simulation enables larger-scale *HIL* evaluation setups. Ding et al. [124] present a cognitive radio network testbed based on parallel-coupled protocol stack simulations, configurable *RF* front-ends, and proprietary channel emulation hardware[2]. In a short demo abstract without evaluation, Klingler et al. [146] present the concept of a bridge based coupling to *IEEE* 802.11p

[2] *RF*nest™ is a hardware channel emulator (class A), developed by Intelligent Automation, Inc.

router hardware connected via *LAN*. They consider real but uncontrolled radio communication to a single device under test via the so called *LAN Radio* (proprietary gateway router hardware with open-source configuration software to process radio transmissions). This approach is similar to [145], but difficult to compare because of missing evaluation. In contrast, the *Split-Protocol-Stack* approach allows for an accurate *PHY* representation of all nodes on a real hardware *NET*. Similar approaches that use external hardware to control the radio packet transmissions on the channel from virtually modeled protocols for each node are not found.

The listed approaches in Table 4.2 are additionally analyzed regarding the *Split-Protocol-Stack* methodology to compare the properties and illustrate the methodological diversity which can be seen at a glance in Figure 4.4[3]. Due to the early development stage and lack of details and evaluations, we cannot include the approach of Klingler et al. [146] in this comparison, but the similarities to [145] potentially lead to similar results.

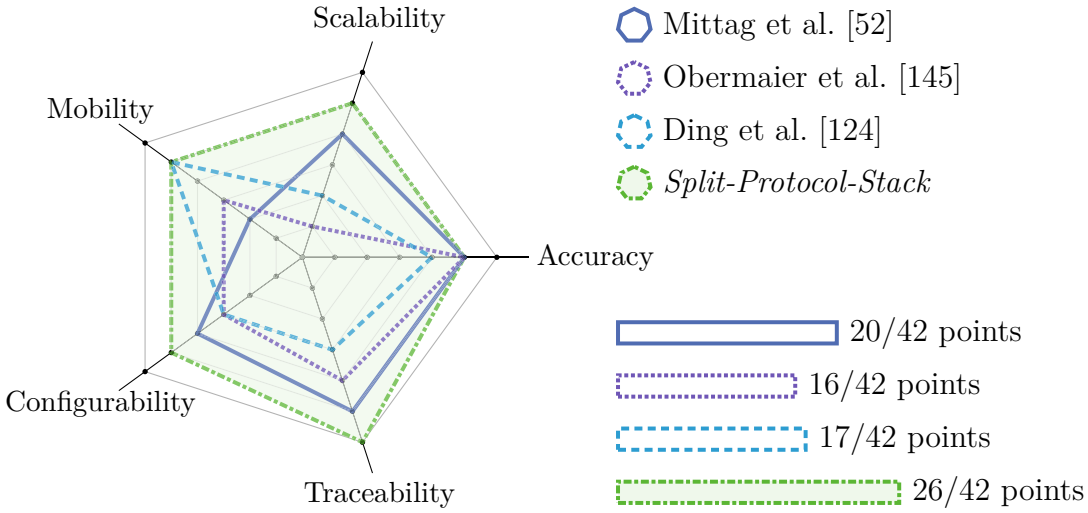


Figure 4.4: Overview analysis of parallel *HIL* and co-simulation approaches running simulated protocols on real or emulated networks based on the properties performance level in comparison with and highlighted for the *Split-Protocol-Stack*[3]. The overall performance is additionally represented by a score.

The virtual signal simulation in [52] can achieve a better scalability (up to 100 nodes), compared to [145] or [124]. Looking at the evaluation in [52], it can be stated that they achieve a much higher accuracy of the radio link, particularly similar performance results as on real hardware chipsets, than traditional packet-level simulation models. Furthermore, the runtime analysis in [52] indicate that most of the computational overhead (approx. 50%) is due to complex channel models in a pure virtual environment, even without node mobility. This problem reminds of what was already discussed for pure discrete

[3] The exact interpretation of the radar plot evaluation is given in Table A.4 in Appendix A

event simulation in Section 2.4. In contrast to the channel emulator use in [124], mobility emulation capabilities in [145] are limited to the virtual nodes (the radio link to the real device under test remains the same). The configurability in [124] is limited with respect to the simulator, as it does not seem to provide a variety of protocol models compared to *OMNeT++*/INET or *ns-2/ns-3*. Due to pseudo-real-time synchronization and real-time hardware-based channel emulation, the *Split-Protocol-Stack* can achieve high accuracy and scalability with attenuation-based mobility for all nodes. Furthermore, the consistent interfacing for the modular general-purpose concept is particularly configurable and enables high traceability based on system events in all evaluation levels.

4.3.3 | Analysis Summary and Concluding Discussion

After the basic strategies and performance-related properties of the selected contributions have been contrasted in a comparative overview, we summarize the state-of-the-art in parallel simulation and emulation to highlight the *Split-Protocol-Stack* approach and discuss the compliance to the imposed requirements in Section 4.1. Of course, all co-simulation/emulation or *HIL* approaches try to overcome some of the limitations of the basic methods (cp. Section 2.4 and Section 3.4), but the methodological diversity highlights the purpose-based tool development and eventually the need for a universal approach. Due to the necessary categorization, a direct and overall comparison of all approaches is not very promising, but problems and requirements (cp. Subsection 4.1.2) as well as some properties can be discussed beyond these boundaries.

What is striking is that related systems either focus on protocol emulation including real software implementations in simulations or they consider real transmissions, but without the *PHY* layer or higher layer modeling flexibility. In our experience, the reasons for this particularly lie in the different views of the communication systems and communication engineering communities. Network-wide performance implications are covered by the protocol algorithms and data frame exchange of the nodes in a network simulation, whereby *PHY*- and radio-channel-oriented research studies primarily address signal processing and propagation of point-to-point connections in practical test setups. The combination of the two research areas potentially creates new synergies that lead to high credibility in evaluating the results of modern wireless research.

Usually, dedicated and partially proprietary components are used, which complicate the configurability and thus exclude a universal evaluation strategy. When looking at approaches for wireless sensor networks, the nodes' operating system is often in particular focus (e.g., *TinyOS* in [136, 139, 142]) On the other hand, not only proprietary host devices and stacks are set up according to *IEEE* 802.11 *WLAN* in [140] and [138], but modular *SDR* components are used as well (e.g., in [124, 145]). Furthermore, Figures 4.3 and 4.4 highlight that real-time hardware-coupled simulations do not enable large-scale network scenarios. As expected, the common simulators *OMNeT++* and *ns-2/ns-3* are

mainly used (e.g., *OMNeT++* in [136, 138, 141, 143, 145] and *ns-2/ns-3* in [52, 139, 140]).

What also can be figured out in general is that *HIL* approaches with connected real external hardware components usually verticaly split the node network, i.e., some nodes are pure virtual and some of them are real. Often only one single *bridge*, *gateway*, *border*, or *proxy* node serves as an interface to a real network which enables to extend a small testbed to a large-scale hybrid network (cp. [145], [146], or [147]). Unterschütz et al. [141] show the splitting of a testbed network in two sets with several gateways, extended by a large-scale intermediate network simulated in *OMNeT++*. It does not aim to increase the radio channel accuracy, but extending a network for certain evaluation questions with reduced accuracy. Finally we can recognize a huge trade-off between scalability, accuracy, and flexibility in parallel simulation and emulation.

Interfacing and Hardware Abstraction

In systems with simulated hardware, an instructions-set simulator can provide clock-accurate communication interfaces, depending on the microcontroller architecture (cp. [142] or [136]). As all components virtually run on the same host system, they use host-based *Inter Process Communication (IPC)* for the inter-simulator communication. The authors of [136] describe their concept as an *asynchronous remote method invocation* with additional message queues for in- and outgoing traffic. Since these systems needs interfaces for cross-domain data exchange, they have to convert between functional-level network and cycle-level hardware simulation events. In [142] the data packet bits from the processor simulator are stored in a transmit *FIFO* of the radio chip module. As soon as the radio chip receives the send command, an event converter transforms the simulation time and sends the packet to the simulated channel. The direct access to the network stack with *Packet Capture (PCAP)* system libraries is also an important method for connectivity within fully virtualized systems when the virtual hardware runs on the same host system as the network simulation (cp. [148] or [128]).

Virtualization of network hardware with virtual device-driver-enabled interfacing is presented in [140] and [137]. Weingärtner et al. [140] implement a *gateway node* which bridges the logic of the *ns-2/ns-3 IEEE* 802.11 model with the device driver. Thus, they are able to receive raw data frames from the virtual network adapter which is used by real applications. Virtual interfaces can potentially be provided for different devices, enabling different configurations, protocol stacks and application implementations for both wired and wireless technologies and network setups. Interfacing a real device unter test via the radio channel in [145] is based on the *physical twin*, a proxy between a simulated and a *SDR*-based *IEEE* 802.11 interface. Simulation events of *MAC* data frames are translated into *over-the-air* messages, while the *over-the-air* interface is a global *OMNeT++* module, handling the *MAC* data exchange with the *Application Programming Interface (API)* of

a well-known but specific *SDR* hardware platform (*USRP*). Thus, it is a very specific setup with a very low hardware device abstraction level which cannot serve for a general methodology.

The interfacing of external devices with the simulator is usually based on the standard host operating system sockets (e.g., raw *IP*, datagram, or stream sockets), but real, resource-constraint wireless device hardware usually does not support sockets and has to be attached via lower layer interfaces. A connection to an external hardware node is achieved by a gateway software, which forwards the data from a socket to the nodes wired communication interface. This is either based on a testbed *IEEE* 802.3 Ethernet infrastructure [124, 138, 143, 146] or simple serial interfaces, e.g., an *UART*, for resource-constrained devices (cp. the transparent forwarder in [141]). Thus, contributions featuring external radio device hardware implement additional bridge software to connect via transport layer socket *IPC* to the simulator. The use of standard hardware platforms as a gateway (e.g., a *Raspberry Pi* in [143]) offers generally a good opportunity to interconnect different wireless hardware in a uniform way.

Synchronization

When running a real-time simulation, the scalability of the setup is limited by the system performance as discussed in Section 2.4 and Section 3.4. Since, pure virtual parallel simulation and emulation do not need to run in real-time, approaches can deploy parallel *DES* synchronization methods which satisfy the *local causality constraint* [129]. E.g., in [136] synchronization is required regarding a packet transmission on the wireless channel in *OMNeT++*. Thus, the overall simulation progress is controlled by the hardware simulation and proceed when an emulated node starts a packet transmission on the channel. Zhang et al. [142] alternate the execution of both engines (network simulator and cycle-level node simulator) by comparing the timestamps of the next execution step or event message. Because both simulators are running on the same host system, they can use the same clock source for fetching the next event timestamp and execute the earliest. Since the *PHY* simulation and virtual channel models are used directly in *ns-2/ns-3* with a library, [52] do not need any event synchronization.

Considering the cooperation with real components in simulation setups, some of the approaches run a real-time scheduler in the simulation. They do not consider or even do not mention any further synchronization (e.g., [124, 143, 146]) which eventually yields to real-time violations. Kato et al. [137] discuss the challenging communication overhead and real-time violations, but a time synchronization is not implemented. Also Obermaier et al. [145] discuss that delayed events from the *SDR* introduce real-time losses and real-time violations are unavoidable with a high amount of external events. Nevertheless, besides the thread-safe real-time event scheduling no solution is presented for this issue. The only approach in our analysis that enables scalable hybrid setups with real components

is *SliceTime* introduced by Weingärtner et al. [140, 149]. They use equal time slices for an alternating execution of real nodes and the simulator with synchronization barriers, similar to parallel *DES* synchronization methods. This is possible because the real-time execution of the nodes higher layer protocols are not influence each other, compared to a *PHY* emulation where several nodes need true parallel radio access. The very similar emulation approach presented by Staub et al. [138] also refer to the *SliceTime* synchronization in consequence to their observed real-time simulation issues. In [141], it is argued that higher-layer protocols used in a hybrid testbed should not rely on accurate timing constraints due to the high latency between simulated and real networks and the ease socket synchronization within the single-threaded simulator. The latency measurements in [141] show that accurate real-time simulation is not possible with hybrid testbeds.

Resource Allocation and Scenario Configuration

When configuring a virtual scenario with a simulated network, the allocation of nodes can simply instantiate virtual components based on the number of simulated devices in the network (e.g., virtual device driver instances in [137, 140], or hardware simulator instances in [136, 142]). Allocation of external hardware nodes is mostly done manually, since scalable scenarios with multiple *HIL* devices are not very common. There are no statements of a testbed node allocation procedure, but very small-scale and simple test setups in [124, 141, 143, 145, 146]. For larger-scale scenarios with real nodes, the allocation and assignment of radio resources becomes a state-of-the-art research issue.

Most approaches do not detail their scenario configuration procedure. This is why it can be assumed that the scenario configuration is mainly done manually. Due to the purpose-bound development background, mostly only specific scenarios with small networks are set up (e.g., single communication links in [52, 143, 145]). The described larger-scale evaluation scenarios present simplified topologies (e.g., placing sensor nodes in a straight line in [52, 139], nodes in a fixed grid in [136], or a peer-to-peer network with a straight line backbone and multiple hosts on each access router in [144]). A randomly distributed network with paired nodes, a ring, star, and tree topology is used to evaluate the approach in [142]. Mobility support is usually limited to model-based virtual mobility. In [124] the nodes can be moved manually or automatically by random mobility. Finally, no general suggestions for the scenario configuration are made within the analyzed approaches, high configuration efforts are also addressed, e.g., in [141].

4.4 | Benefits, Use Cases and Scenarios

Parallel network simulation and emulation combines evaluation techniques for different disciplines of wireless communication expertise, e.g., software development, protocol design, radio channel propagation effects, radio frequency physics, and hardware development. Approaches featuring *HIL* techniques for the link layer are common. At higher layers, these techniques are rarely deployed [39, pp. 120ff].

4.4.1 | Approach Benefits Overview

With the *Split-Protocol-Stack* approach, we are able to execute the entire protocol stack in a configurable radio link hardware accurate laboratory setting for realistic protocol and network performance evaluation with state-of-the-art analysis tools. As a novelty, the behavior of new protocol procedures for wireless access technologies in a real radio channel network setting can be investigated, instead applying abstracted propagation models for the wireless transmissions which helps to increase confidence in the protocol simulation-based network evaluation. This allows simulation-generated packet data to be transmitted over real radio data packets on the radio channel. As a result of our related work analysis, some of the benefits of combining protocol simulation and radio link emulation with our evaluation approach are summarized shortly below and illustrated using various use-cases and configuration scenarios.

- **Radio link accuracy** is often considered as very limited in pure network simulations as discrete event simulation cannot model the dependencies, characteristics and parameters of the radio transceivers. With *RIL*, these issue can be eliminated by improving the accuracy of the *PHY* layer and radio simulation models. Thus, the overall credibility of simulation-based performance results can be increased.
- **Real-time scalability** can be increased because of real-time channel transmissions, since propagation models are very resource-expensive.
- **Traceability** of network and node states, processes, and events reaches a very high level, as all protocol-related inter-layer and inter-system interactions as well as several hardware-related properties, e.g., node energy consumption, can be monitored.
- **Scenario configurability** in model-based design is very high, since many system and network components can be adjusted precisely, e.g., network topology, protocol arrangement, algorithms, applications, and traffic patterns simultaneously to *PHY* parameters, procedures, and the environmental radio settings.
- **Node mobility** is still model-based or from scratch, but has a real impact on the retrievable channel state information because the attenuation between real nodes will also be affected.

4.4.2 | Application Domains and Use Cases

The *Split-Protocol-Stack* emulation methodology ultimately represents a proposal for a modern evaluation and research platform for opportunistic and future wireless networks when conservative approaches reach their limits. In the following, we point out some state-of-the-art use-cases where we expect a particular benefit.

(a) Cross-Layer Optimization in Cognitive Radio Networks

Based on our analysis, we argue that the functional requirements for accurate cross-layer optimization and cognitive radio evaluation can be fulfilled by parallel simulation and emulation. For example, the authors of [123] present a list of requirements for prototyping the *MAC* layer. Among other things, they refer to access to accurate *PHY* information and the need for *PHY* reconfiguration based on the *MAC*. They conclude that mixed hardware/software architectures of embedded *SDR* platforms are the best choice to provide full flexibility and reconfigurability. In cognitive radio networks also the higher protocol layers are effected by the spectrum management and need reconfiguration based on the *PHY* sensing information and the radio access scheduling. Thus, cross-layer optimization plays a crucial role in cognitive radio networks and can involve all layers of the communication protocol stack (cp. [150] and [20] for the information exchange based on cross-layer interaction). With the *Split-Protocol-Stack* approach, detailed channel state information can be obtained from an accurate emulated channel and *PHY* parameters can precisely be adjusted on real transceiver hardware or completely reconfigured based on *SDR* modules instead of applying abstract information for packet data in pure discrete event simulation [50]. In addition, our methodology offers great research and prototyping opportunities regarding cross-layer optimization of the energy efficiency.

(b) Network Robustness and Resilience in Dense Radio Environments

In network simulations dense radio environments with a huge amount of transmitting nodes are unfeasible because the calculation complexity causes high computing demands and runtime (cp. analysis results of [144] in Section 4.3 in which large-scale scenarios with radio link simulation models are 15 times slower than in real-time). Furthermore, modeling co- and inter-channel interference incurs a high overhead in addition and is a feature that is often missing in current simulators [44]. On the *Split-Protocol-Stack NET*, hundreds of nodes can be set up to form multiple networks on neighbored channels (potentially with different radio link technologies) in real-time. Dense radio environments can also be extended with a *SDR*-based hardware to introduce specialized *PHY* features, e.g., high noise levels or even frequency-selective blocking and jamming signals. Protocols and algorithms that ensure robustness and resilience with intelligent methods at the higher layers can be analyzed in the simulation.

(c) Heterogeneous Wireless Networks, Technology Coexistence and Interoperability

Usually, simulation studies focus on homogeneous wireless networks which share the same technology and radio channel resources. In real world, however, wireless networks are rarely homogeneous, since various radio technologies are deployed, especially in the unlicensed *ISM* frequency bands. Especially in the field of industrial *IoT*, there are many vendor devices and protocol stacks that use different access technologies but affect each other. Often they even have to take over the control tasks in interworking network architectures, where, for instance, mechanisms for prioritizing communications become necessary (cp. [151]). But even in a homogeneous setup, cooperative spectrum sharing is needed in parallel networks. The *Split-Protocol-Stack* approach provides an excellent environment for testing cooperating protocol and network architectures on heterogeneous and cooperative wireless access networks.

(d) Wireless Internet of Things Perception Layer Security

The *Perception Layer* [152] unifies the *PHY* from a communication protocol point of view with all other sensors for capturing and gathering information in the nodes' environment. Studying security vulnerabilities based on the perception layer in real wireless *IoT* networks needs physical access to the radio channel and to the physical sensing environment. It is therefore usually performed in testbeds. Thus, a specific test setup with node firmware is required for each vulnerability or demonstrated attack. The *RIL* bridge node can be used as a sniffer device for only frame reception and monitoring or for demonstrating protocol layer vulnerabilities to study the impacts of injecting radio frames into a real network. Thus, radio link attacks, such as jamming, tampering, exhaustion, collisions, or higher layer protocol attacks, e.g., replay, flooding, spoofing, malicious data, and many more, are feasible with great model-based flexibility for the insertion of malformed or modified radio communication. For example, a simulated attack network scenario from well-established internet protocols can result in attacking a real-world wireless *IoT* network in a testbed which can be analyzed in detail (cp. [153] for an exhaustive survey on perception layer security and protocol layer based attacks).

(e) Machine Learning for Physical Layer Communication and Wireless Networks

Machine Learning (ML) strategies for *PHY* wireless communication are a red-hot research area with potential synergies with advanced link layer technologies. Learning from channel state information and beyond [154] [155] [156] needs real-world data sets or even physical real networks in a testbed for the learning phase. On the other hand, discrete event simulation can easily integrate algorithms besides the communication protocol stack for studying possible use cases. Thus, the *Split-Protocol-Stack* is a perfect basis for integrating channel state information into the simulation of machine learning techniques.

4.4.3 | Split-Protocol-Stack Reference Configuration Scenarios

From a methodology perspective regarding the protocol stack, there are three basic kinds of nodes: *simulated*, *Split-Protocol-Stack emulated*, and *real* ones. Simulated nodes are entirely virtual. They consist of simulation modules for all system components and the communication protocol stack, whereas real nodes are implemented holistically in hardware and firmware. *Split-Protocol-Stack* emulated nodes are composed according to the methodology presented in Section 4.2 which in addition to simulated higher-layer protocols provide a real transceiver hardware and a *PHY* implementation capable of transmitting real packet data on an emulated channel via the *RIL* technique.

From an application point of view, the nodes can of course provide diverse device types, services, and functions and thus various protocol stacks and link layer technologies. These nodes can be realized by means of the three basic types in the scenario configuration. In particular, this means, e.g., a network coordinator or end device according to *IEEE 802.15.4* can exist physically real, hybrid, or purely virtual in the application scenario. With the consideration of the approach benefits, an *internet gateway* according to *IETF*, for example, is more likely to be implemented in the simulation domain, whereas sensor devices are more likely to be set up with real nodes.

Considering the evaluation use cases mentioned above from a protocol stack's point of view (as introduced in Subsection 2.1.2), we present two different generic configuration schemes as a basis for purpose-specific application scenarios in the following. In doing so, we illustrate the integration of the *Split-Protocol-Stack* in a simulation-driven evaluation system instead of constructing application-specific network topologies. This scenarios simultaneously serve as reference configurations for the exemplary system and prototype evaluation in the subsequent chapters.

Pure Wireless Link Emulated Networks

Considering the whole setup as a homogeneous wireless network (e.g., a *WSN*), all nodes radio communication is emulated on the hardware testbed. These can either have completely static real transceiver hardware and implementations or be modeled using the *RIL* approach. The latter is referred to below as *emulated* or *Split-Protocol-Stack* nodes and can be analyzed in the simulation regarding the communication with the *PHY* and all protocols of the higher layers. Real nodes do not offer these insight analysis properties, only the external radio communication can be monitored. Emulated nodes can be used to create simulation-based network topologies for the testbed that are otherwise elaborate to reproduce. Figure 4.5 shows a generic configuration scenario with purely emulated radio links in which the number of real and emulated nodes can be adjusted individually. Note that the view of the nodes' protocol stack does not show a specific topology, but only several nodes that are part of the same channel emulation and can form different radio topologies, as introduced in Section 3.2.

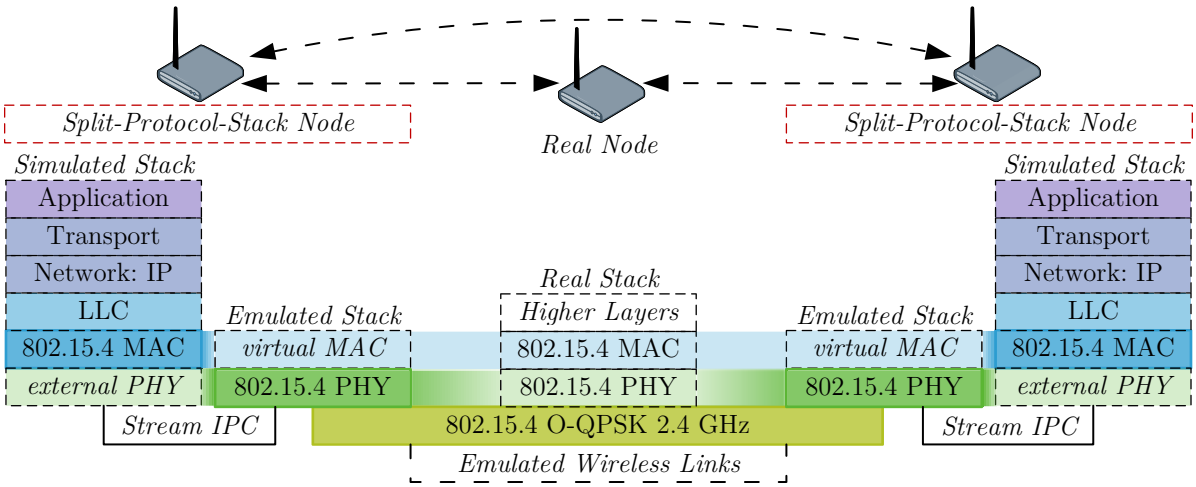


Figure 4.5: A configuration scenario for pure wireless link emulated networks. Nodes can be either completely real at hardware and firmware or simulated at higher layer protocols with the *Split-Protocol-Stack*.

This configuration scenario is fundamental to create dense radio environments for the analysis of protocols and applications based on emulated nodes for robust networks (cp. 4.4.2b). In addition, it serves as an excellent basis for studying cross-layer optimization strategies for cognitive radio networks (ref. 4.4.2a), as well as for machine learning strategies embedded in the protocol stack (ref. 4.4.2e).

Hybrid Gateway Networks

A gateway configuration scenario extends the pure emulated wireless network with other nodes and network components in the pure virtual simulation domain to form a hybrid network. This can be either wired network devices or wireless nodes with model-based virtual channel communication and interfacing. A *protocol gateway* node is equipped with a minimum of two parallel network stacks because it translates from one communication protocol to another. It can operate on all layers of the *OSI* reference model or the *TCP/IP* protocol stack. Considering a *Split-Protocol-Stack* node, a gateway at least has to translate to and from the *RIL* external node. With the focus of this thesis, we consider *layer 2* and *layer 3 (router)* emulation gateway nodes. Higher layer gateways can be modeled based on the same scheme, but because of the all-embracing *IP*, a translation of the network layer is usually not necessary. Figure 4.6 shows two configuration scenarios with a *Split-Protocol-Stack* gateway each. A router (layer 3 gateway) node connecting to a different technology virtual wired network (e.g., *IEEE* 802.3 Ethernet) is abstracted in Figure 4.6a. It shows a configuration scenario with a bridge (layer 2 gateway) node connecting to a same link layer technology (*IEEE 802.15.4 PHY*) virtual wireless network. Of course, these hybrid network configuration scenarios can also include additional basic *Split-Protocol-Stack* nodes, as depicted in Figure 4.5.

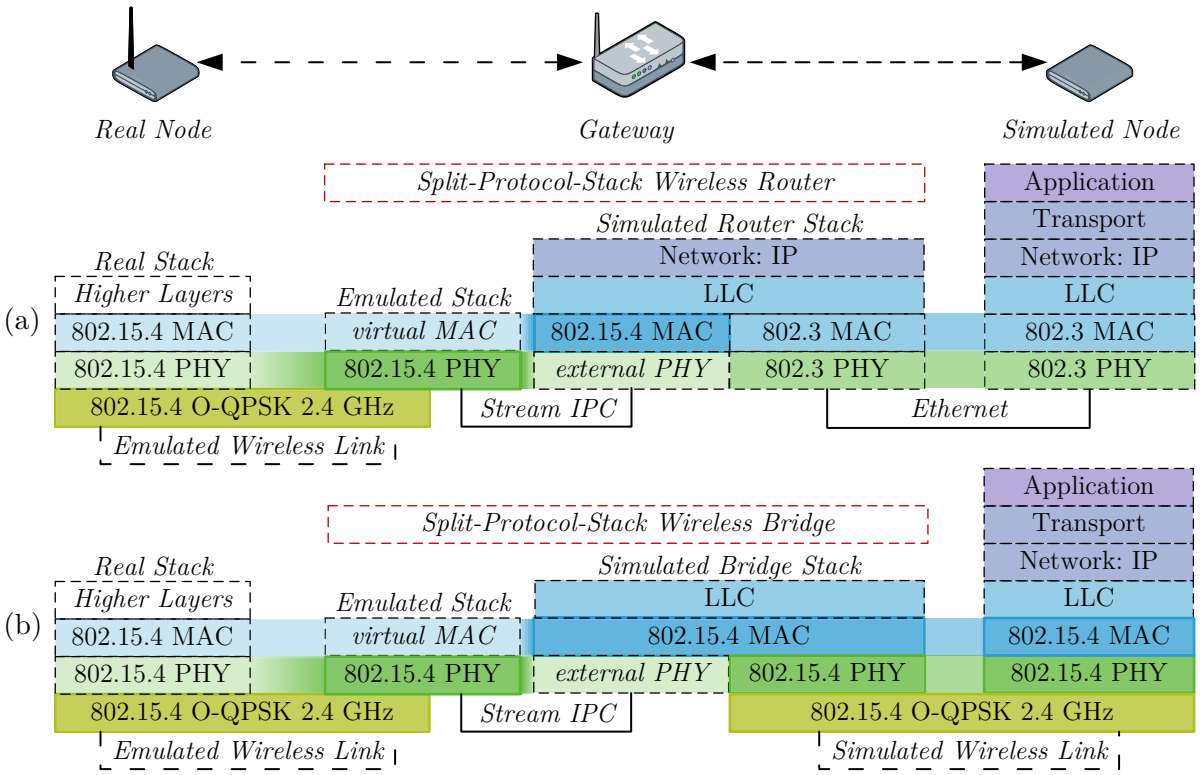


Figure 4.6: Two configuration scenarios for hybrid gateway networks. **(a)** shows a router (layer 3) connected network to a Ethernet-based virtual wired standard computer network. **(b)** shows a bridge (layer 2) connected network to a 802.15.4-based virtual wireless sensor network.

Since star networks are still the most widely used topologies for wireless networks in practice, often only one central node of the scenario is emulated as a gateway that connects to a larger (wired or wireless) control network, while other nodes are running sense and control applications. Extending a radio channel emulated network with a virtual control network is the recommended configuration scenario, as it serves as a perfect basis for wireless *IoT* perception layer security research (ref. 4.4.2d) or heterogeneous wireless networks and technology interoperability (ref. 4.4.2c). Inside the simulation environment, for example, a larger network attack scenario can be prepared, which eventually results in attacking or disturbing a real network on the testbed. On the other hand, a heterogeneous network with interoperable technologies usually needs a backbone control network for the management and configuration, which can be perfectly modeled within the network simulation domain. These two suggestions usually rely on standard network components (e.g., servers, routers, switches) with their appropriate links in arbitrary network topologies for which common simulation frameworks provide an extensive model library.

5 | Real-Time-Shift Discrete Event Simulation & Synchronization

With the analysis of parallel simulation and emulation approaches in Chapter 4, it can be stated that there are no synchronization methods for simulating protocols on real or emulated networks. This is because of the limitations when running real-time simulations. Related synchronization approaches for running real or emulated protocol implementations in simulated networks, e.g., *SliceTime* [144], cannot be applied here because the emulation of the *PHY* and the radio channel in a cooperative wireless network needs true parallel radio access of the nodes. With the *Real-Time-Shift* synchronization, we contribute with an approach that decouples the simulator from real-time constraints by introducing a *pseudo-real-time* simulation [57].

In the following, we give a detailed overview of the addressed synchronization problem with regard to related work in *PDES* and co-simulation. From the findings of this analysis, we derive our solution approach and present the conceptual design of the *Real-Time-Shift* pseudo-real-time synchronization between simulated protocols on an emulated network. In addition, we give important details on the concep implementation, the interface definitions, and the *IEEE 802.15.4* model-interaction in *OMNeT++*. The chapter concludes with an evaluation regarding the feasibility, a performance parameter study, and a discussion of the achievements.

5.1 | Real-Time Synchronization Problem Statement

Communication between two layers of the same system is very fast and complete in contrast to a communication between two systems over their network interfaces. Because the link layer protocols are usually implemented in a single embedded network interface of a standard specification (cp. link layer technologies in Section 1.1, e.g., *IEEE 802.15.4*, *Z-Wave*, or *LoRaWAN*), the communication time between the *MAC* and *PHY* can be compared to a system call or a simple *Local Procedure Call (LPC)*. When running the simulation and real hardware components in the emulation domain in parallel with the *Split-Protocol-Stack*, additional time for the transmission of the *external* and *emulated* event messages is needed. Thus, the communication between *MAC* and *PHY* needs *IPC* on the same system or even *Remote Procedure Call (RPC)* between fundamentally different host systems via network communication. Since link layer specifications define strict constraints and thresholds to enable a time-accurate medium access, this overhead

in time will violate the *MAC* procedures. For example, in order to comply to the fixed time intervals between radio packets, called *Inter Frame Spacings (IFSs)*, the *PHY* has to process the *MAC* instructions just in time. This means in particular for a *RIL* setup: when a *request* is transmitted with a time overhead to the real hardware, the *response* is always delayed.

Behind the scenes of distributed and parallel *DES*, there are two ordering strategies [129] for the execution of external events: the *receive-order* (1), and the *timestamp-order* (2). When applying the first type, events are scheduled immediately after arriving at the destination process. This might be too late or too early due to the varying transmission latency (jitter). With the timestamp-order, events can be put into a sorted queue before execution, but they need to be created and buffered before reaching their schedule time.

5.1.1 | Real-Time Hardware-in-the-Loop Simulation

Real-time *HIL* simulation approaches usually use the receive-order strategy (1) with no further synchronization of event messages, as observed in Section 4.3 (cp. e.g., [141], [143], or [145]). This is only possible when the processing of an event can be handled until the occurrence of the next event, as mentioned with the *DES* basics, depicted in Figure 2.5. In a hybrid simulation and real-time emulation setup it must be ensured that both the scheduling of the *emulated* events to be transmitted on the real hardware and the scheduling of the *external* events in the simulation are possible. Since simulation overload and event execution delays consequently lead to real-time violations, often applied approaches limit the number of simulated events by scaling-down the scenario, enlarging the real-time threshold, or powering-up the simulation core with sufficient computational resources for virtual event processing (e.g., the number of simulated nodes in [145] is limited to 3 while a high threshold of 100ms is accepted). These strategies are not feasible when a microseconds timing accuracy in large-scale network scenarios is needed.

To prevent event dependency violations, time-accurate scheduling based on a timestamp-order (2) of the events is required when running multiple interdependent simulation and emulation processes in parallel. Since usually a *DES* core runs in a single thread, each simulation step is required to be synchronized with the event scheduler (ref. Algorithm 1 in Subsection 2.1.3). This means, that the simulation cannot proceed with the execution of internal events until a dependent emulation event has finished. In the research domain of *PDES* two classes of synchronization algorithms are distinguished to guarantee the compliance with the causal order, *conservative* and *optimistic* synchronization schemes. In optimistic schemes events are processed speculatively during the previous computation and are rolled back in case of synchronization errors. A hardware-based channel emulation cannot be rolled back as in optimistic interactive simulations because the hardware is running in real-time. Conservative schemes, on the other hand, guarantee an error-free

execution without the need for a roll-back mechanism. These approaches, however, are only suitable for non-real-time parallel and distributed simulations, as they block the simulation process from further processing until it can be assured that the next event in a simulator's *Future Event List (FEL)* can be executed before any external event arrives in the future. Therefore, they often require a look-ahead about the future behavior of a connected system which is not possible with real-world systems.

This has been already figured out by Weingärtner et al. [140] when coupling real nodes´ network stacks on *Virtual Machines (VMs)* with a *PHY* network simulation. The methodology of *SliceTime* [144] consequently decouples the overall simulation time from the runtime using a synchronization scheme similar to the *Conservative Time Windows (CTWs)*. It allows every attached peer to execute for a certain amount of time, a so-called *time slice*, until a synchronization barrier is reached by all other peers. Thus, the synchronization accuracy is given by the size of the time slice, (e.g., sufficient in the range between 0.1ms and 2ms), whereby smaller time slices (higher synchronization accuracy) directly yield to a higher accuracy of network measurements (cp. latency measurements in [144]). A global *synchronizer* component is responsible for signaling the time slices to all connected peers. With small time slices, there is a large amount of messages that need to be exchanged (e.g., a time slice of 0.1ms needs to issue 10000 time slices to all *VMs* each logical time second). The synchronization overhead compared to a real-time execution is comparatively low for a single *VM*, but increases linear when incorporating multiple nodes (e.g., emulating 20 nodes, need approx. 20 times of the real-time). Running *CTWs* with *SliceTime* is possible when running higher layer protocols of multiple nodes which do not influence each other, but is not suitable when running a *PHY* emulation in which multiple nodes need parallel radio access. The difficulties occur when many participants in the network want to access the medium or even start it at the same time.

5.1.2 | Real-Time Synchronizing Simulation and Emulation

When running multiple subsystems in parallel while relying on timestamp-order for the event execution, all peers need to synchronize with a global clock source to calculate a common time base in addition to the aforementioned event synchronization. On the same host system, the global time can simply be given by a precise operating system timer, whereas it has to be assured in distributed systems that the clocks of all systems are synchronized precisely. For standard host systems, the deviation from the global exact time can, of course, be decreased by clock-synchronization for which well-known network protocols (e.g., the *Network Time Protocol (NTP)*, or *PTP* on dedicated *LAN* for high accuracy clock synchronization) still exist. However, especially on resource-constrained sensor node hardware, the oscillators for the system clocks are usually particularly imprecise and interfaces and implementations for network clock synchronization are missing. For example, on typical wireless sensor nodes it is about $10 - 100$ppm (thus after 1h operating time, the deviation is between $40 - 400$ms).

5.2 | Real-Time-Shift Network Simulation

When analyzing a *Split-Protocol-Stack* parallel protocol simulation and network emulation, only the access to the radio channel in the emulation domain has to be executed in synchronized real-time. With each radio access of a single node, all nodes need to be in their currently assigned state. On the other hand, when only *virtual* events occur in a certain time, there is no need to care about the emulation domain. Therefore, we rely on an alternating execution of the protocol stack operations and the radio communication.

We propose *Real-Time-Shift* pseudo-real-time synchronization as solution, a *conservative* scheme to strictly avoid the occurrence of causality errors and ensure real-time execution of the *PHY emulated* events. Pseudo means that the overall evaluation process runs at real-time, but the simulated time is stretched by waiting times during the transmission of emulated and external events. This waiting times have to be compensated by the simulated time to stick to the actual real-time. Accordingly, the (virtual) simulation time is permanently shifted back and forth by the scheduler due to the event transmission delays. In the following subsections we introduce the fundamental concept of event transmission latency and jitter compensation, the pseudo-real-time scheduling approach within the *DES*, and the real-time stream processing to the *RIL* hardware.

5.2.1 | Event Transmission Latency and Jitter Compensation

Besides the overall evaluation runtime (wall time), there are three different observable times: the *simulated* real-time, the *emulated* real-time, and the emulated/external *event transmission* time. The emulated real-time is defined as the sum of the event execution times on the real hardware *PHY* processing and transmission on the radio channel. The emulated/external event transmission time is limited by an upper boundary and the simulated real-time is the overall evaluation runtime minus the emulated/external event transmission time. The time that elapses during the transmission between the involved systems is neither constant nor can be exactly predicted before sending. Inevitably, network transmissions introduce a variance in end-to-end latency, called *jitter*. The latency and jitter depends both on the event message length, which is comparably small and has an upper limit, and on the underlying control network in the emulation domain. Within a dedicated *LAN* or on a single host system, an upper boundary for the transmission time can be determined. To compensate event transmission latencies this upper boundary is needed to define the fundamental pause time interval or maximum transmission latency (τ_L).

The Figure 5.1 shows a real-time simulation setup with compensated event transmission latencies. The events e_1, e_2, and e_5 are pure virtual, while e_3 represents an emulated *request* and e_4 the respective external *response* from the physical hardware. The event processing time of the emulated event (T_{e_i}) represents the real execution on the target

hardware, e.g., the time to perform a channel switching. The pause time (τ_L) defines the determined maximum amount of time (*event transmission time* in Figure 5.1) when transmitting emulated and external events. The actual transmission time is designated with (L_{e_i}) in Figure 5.1. With every (emulated) *request* event message from the *MAC*, the *PHY* generates a timestamped (external) *response* event. Thus, the execution time of the request (*event emulation time* (T_{e_3}) in Figure 5.1) on the physical hardware can be calculated in the simulation. The real hardware execution and event compensation remain invisibly in the simulated time (removing the pause times).

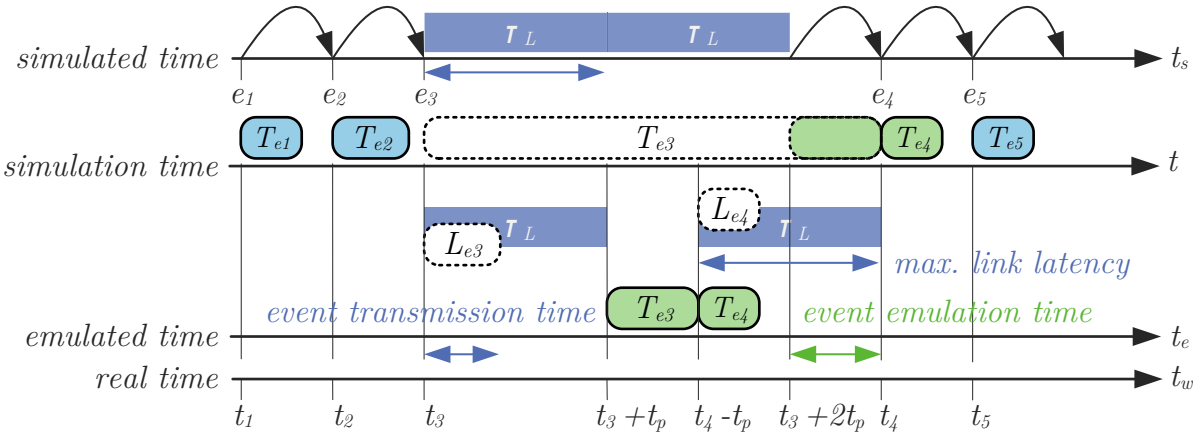


Figure 5.1: Time compensation in real-time event execution (cp. basic real-time simulation in Figure 2.5)

5.2.2 | Pseudo-Real-Time Event Scheduling

A pseudo-real-time scheduler in a *DES* must take the above mentioned time compensation into account when running the event routine on the *FES*. Our solution here is an extension of pure real-time *DES* (ref. Section 2.1.3) to enable the accurate scheduling on multiple interface-connected real hardware systems, e.g., *RIL* transceiver. Depending on the number of *RIL* nodes in the emulation domain, all relevant events of a time period in case of concurrent media access have to be distributed to all involved hardware nodes with exact timing. To achieve this, some fundamental assumptions have to be made:

(a) The simulation must be real-time capable and all events must be scheduled in real-time (ref. real-time *DES* in Subsection 2.1.3),

(b) The simulator must be able to interrupt and resume the real-time event processing without terminating the simulation,

(c) The simulator must be able to distinguish between event types (*virtual*, *emulated*, and *external*),

(d) The maximum *event emulation time* of a single event on the real hardware must be ensured to be smaller than the *pause time* (τ_L) in the simulation.

The approach is to pause (freeze) the scheduling in the simulation at certain times when violations occur due to latency or long computation cycles. The resulting simulation pause time is subsequently called the *Real-time-Shift Vector (RSV)*, which must be configurable according to the time interval for inter-layer communication in the emulation setup. The clock source of the scheduler is the simulator's host system *wall* clock (t_w), but the overall simulation time base is mentioned the *Global Virtual Time (GVT)*. Since the scheduler is running in real-time speed, the GVT is equal to the elapsed real-time since simulation start minus the number of simulation pauses (n_P) multiplied with the RSV. If there is a simulation start time not equal to zero it must be taken into account also. Since the RSV is equal to the event transmission time or link latency (t_L) multiplied by two, the following equation can be used to calculate the current GVT:

$$GVT(t_s) = t_{s_{Start}} + (t_w - t_{w_{Start}}) - RSV(2\tau_L) \cdot n_P \tag{5.1}$$

n_P – number of simulation pauses
τ_L – max. event transmission time (link latency)
t_s – simulation time (virtual clock)
t_w – real-time (wall clock)

Equation 5.1: Calculation of the GVT based on the link latency τ_L.

If there is only one single emulated *RIL* protocol stack and device the scheduling can be achieved according to the basic latency compensation mentioned above. If there are multiple nodes in the network simulation all *simultaneous* events (occurring within the time of the scheduled *emulated* event) have to be calculated and scheduled also. According to the fundamental assumption in (b), the simulation must be fully responsive during the pause to schedule all concurrent events from competing nodes and receive *external* events as responses to the *emulated* ones.

A concurrent event scheduling and execution based on the fundamental latency and jitter compensation scheme is depicted in Figure 5.2. According to this example scenario the *external* event e_3 is detected by the scheduler which introduces a simulation pausing due to the transmission latency τ_L to the *RIL* hardware. Consequently, the *FES* is looked up to determine the events that will occur during the pause time (τ_L). All other future events are *shifted* in time with the amount of RSV ($2\tau_L$ is added to the timestamps). The simulator proceed with the real-time scheduling and the remaining events within τ_L are scheduled according to their timestamps. Thus, pausing the simulation is done only virtually by manipulating the *FES*, but the simulation scheduler is not interrupted in its execution.

The influence of the RSV on the GVT (t_s) is abstracted at the top of Figure 5.2. Thus, the real-time *emulated* event is shifted with a statical amount of time to be executed on the target hardware. According to the general time compensation scheme the emulation time of each emulated event can be calculated based on the succeeding external event occurrence time. This opens up possibilities for profiling that are not possible in an

ordinary real-time simulation, since when an emulated event is scheduled no reference is made to its processing time. For example, regarding the scenario in Figure 5.2, the execution time of e_3 on real hardware is equal to the difference of e_5 (*response*) minus e_3 (*request*) in GVT.

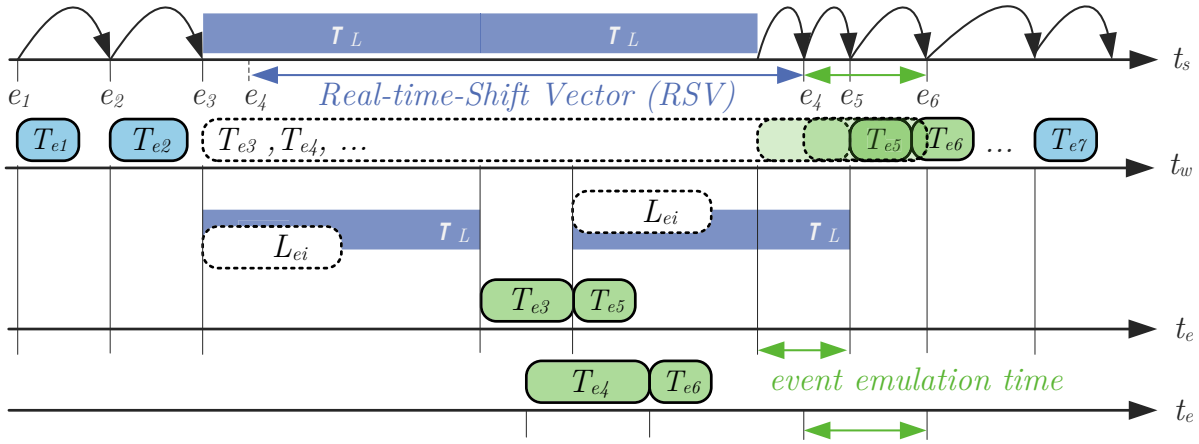


Figure 5.2: Pseudo-real-time concurrent event scheduling within the basic time compensation scheme

In order to enable the scheduler to distinguish between normal event execution and the simulation pause state we introduce two new event types, called

- *simulationPause* – all future events in the *FES* will be delayed with the time amount of the *RSV*, and
- *simulationResume* – the simulation time is shifted back with *RSV* and the before mentioned delayed messages will reset to the original creation timestamp.

The insertion of the *pause* and *resume* events is performed by the simulation model to facilitate event handling within the scheduler. Thus, the *Real-Time-Shift* scheduler only need to handle pause states of the *GVT* and do not need to know about *virtual*, *emulated*, and *external* events.

The procedure of the *Real-Time-Shift* simulation scheduling is shown in Algorithm 2. According to the fundamental *DES* scheduling in a real-time setup, each event is not processed until the event time is reached. During these waiting periods, the scheduler listen for *external* events on the connection interface to the emulation domain. Since a receive is a blocking function call that prevents the single-threaded simulation from processing further code, it should be executed in chunks of a predefined timeout constant $\tau_{T/O}$ (line 6 in Algorithm 2). Between this chunks, the simulator can check for non-scheduling related activity, e.g., interrupts from the simulation user interface. If something appears on the connection interface, an *external* event v is created and inserted in the *FES* within the receive routines. According to the pseudo-real-time strategy, v is inserted

with the time shift offset τ_L and marked as shifted from real-time.[1] This causes v to be executed not before the correct processing time is reached with respect to the offset compensation. During the *RSV* pause time, further external events can be received on the connection interface and no external events will be executed. Thus, the correct next event e will be processed by the corresponding event handler routine.

The indication for the scheduler to release a real-time shift operation is given with an *simulationPause* event as a result of an *emulated* event from the simulation model. Consequently, all events e in the *FES* with a timestamp $(t_s + \tau_L) < t_e < (t_s + 2\tau_L)$ are shifted by that amount for future processing and marked as shifted with the remembered original timestamp (lines 14-15 in Algorithm 2). The already scheduled *pause* and *resume* events are excluded from this shifting. Finally, the event processing is executed analog to the normal *DES* scheduling in Algorithm 1.

5.2.3 | Real-Time Event Stream Processing

As long as considering only the virtual event scheduling within the simulation domain according to the *GVT*, the simulation scheduler can serve for the time-accurate handling, but in order to execute *emulated* events on the *RIL* nodes, events must be transferred across system domains. Because interfaces on the different domains vary, there is a need for intermediate components to decouple the simulator from specifics of the emulation domain. Thus, information about the *GVT* and the event message data must be streamed to the targets.

We introduce a streaming interface between the simulation and the emulation domain for real-time aware processing of *emulated* and *external*, called *Event Stream Forwarder (ESF)* (ref. realization in Section 5.3). During the execution of purely *virtual* events in the simulation, the stream processing at the *ESF* is in a waiting state. Looking at *PDES* implementations, one usually can find a *time control interface* and a *data communication interface*. Within the *Split-Protocol-Stack*, the timing control interface is seamlessly combined with the data communication interface, because each event has its own execution time attached to it.

Due to the stream-based event transmission, the start of the data transmission can be used to initialize the *GVT* from the simulation system. Furthermore, the *GVT* increases with the amount of the *RSV* when the initial *emulated* event is detected by the *ESF* in this time interval. When no *emulated* or *external* events occur in a pure *RIL* nodes scenario, the *ESF* is in an active waiting state and the *GVT* proceeds according to the synchronized wall clock. The question at which point in time *external* events happen automatically arises from the fact that the *PHY* (on the real hardware) cannot continue to run without the *MAC* (the virtual node), as it controls the accesses to the *RIL* nodes.

[1] Note that the event timestamps are always set according to the *GVT*.

Algorithm 2: Real-Time-Shift Event-Scheduling Algorithm

Precondition : Initialize *simulation model*, *FES* **and** *time constants* τ_L, $\tau_{T/O}$

```
(1)  while (FES not empty) and (simulation time limit not reached) do
(2)      fetch first event e from FES
(3)      calculate event target time t_e with t_{s_Start}
(4)      get current simulation time t_s
(5)      if t_e > t_s then
(6)          receive until t_e        ▷ blocking receive in chunks of τ_{T/O}
(7)          if something is received then
(8)              create event v
(9)              if simulation is paused then
(10)                 remember arrival time and mark v as shifted
(11)                 t_v += 2τ_L      ▷ add 2τ_L to the actual timestamp
                     end
(12)                 insert v into FES
(13)                 break
                 end
             end
(14)     if e is pause event then
(15)         for all events e excerpt pause and resume in FES do   ▷ time shift
(16)             if (t_s + τ_L) < t_e < (t_s + 2τ_L) then
(17)                 remember arrival time and mark e as shifted
(18)                 t_e += 2τ_L      ▷ add 2τ_L to the actual timestamp
                     end
                 end
(19)     else if e is resume event then
(20)         for all events e marked as shifted do                  ▷ reset shift
(21)             reset arrival time and remove shifted mark
                 end
(22)         t_s -= 2τ_L          ▷ subtract 2τ_L from simulation time
(23)     else
(24)         process event e              ▷ normal event execution
         end
     end
```

Thus, *external* events can only occur if an *emulated* event for data transmission has been previously executed by the scheduler of the simulation, i.e., only if a node has sent an air frame before another node can receive one.

Clock Synchronization

The execution of a *Split-Protocol-Stack* scenario requires all systems running in a dedicated *LAN* or at the same host system, so that system clocks can be synchronized accurately. Thus, the event processing can assign the respective time stamp directly when the *external* event occurs from the local time or wait with the execution of the *emulated* event until the time stamp of this *request* is reached. This requires knowledge about the current simulation time. If all sensor nodes are of the same type, initial synchrony can be achieved by starting the event processing on the hardware execution in parallel. However, due to effects like *clock drift*, it must be ensured that the execution time also remains synchronous over the entire emulation run.

In order to keep the intermediate system flexible in terms of the used end systems we assume an active time synchronization to decouple the timestamp generation from the intermediate system event scheduling. Thus, the intermediate scheduling system does not need to take the current hardware configuration into account (e.g., switches, controllers, link technologies). *PTP* is assumed for synchronizing the wall time of all involved host systems because the accuracy of the *NTP* protocol is not sufficient for this purpose. Due to our *RIL* modeling strategy in which the time-accurate scheduling is part of the host software (ref. Section 6.1), we do not synchronize the *RIL* transceiver hardware. The *clock drift* on low-power wireless nodes can be neglected.

Offset Real-Time Jitter Compensation

To schedule the events at the accurate real time the amount of τ_L is added to the original event timestamp. According to the basic latency compensation mentioned in Subsection 5.2.1, the event e is processed after a waiting $\tau_L - L_e$. Thus, the processing of the event is shifted with the offset τ_L. Based on our *RIL* strategy, introduced in Section 6.1, we cascade these offset time compensation with the *RIL* host system. Thus, a time-accurate execution at the real hardware can be achieved by a constant transmission rate due to a fixed step speed on dedicated point-to-point transmissions.

5.2.4 | Determination of the Time Constants

According to our fundamental assumption in (b) that the maximum event *emulation time* T_e must be smaller than the *pause time* τ_L, the emulation time of a single event has to be figured out first. The event emulation time heavily depends on the used protocol standard

because it represents the maximum amount of time, a *PHY* operation can take on the real hardware. Due to a radio packet transmission, this is the processing time of a maximal-sized data frame which is mainly affected by the modulation scheme. In addition, also particular *PHY* procedures like the *ED* according to the actual configuration and scenario must be considered as time-consuming. Looking at our reference protocol standard *IEEE 802.15.4*, the exemplary *ED* execution time of 138ms provides a good balance between scan duration and accuracy under coexistence with *IEEE* 802.11b (cp. [157]).

Furthermore, the maximum event *transmission time* L_e between the connected subsystems is an important indicator, but one needs to distinguish between dedicated point-to-point (e.g., *RIL* hardware link) and point-to-multipoint (e.g., simulator to *ESF* link) connections. The data rate on the bottleneck link (e.g., *UART RIL* connection) and the maximum event message frame size is used to determine the maximum point-to-point event transmission time $L_{e_{RIL}}$. Moreover, also the number of *RIL* nodes in the scenario configuration must be considered for the point-to-multipoint event transmission $L_{e_{ESF}}$ from a single-threaded simulator, since multiple emulated events can possibly transmitted within the same pause time τ_L. In addition, there is a time variance of the end-to-end delay J_{L_e} (jitter) which is included twice to achieve a certain reserve. Thus, the pause time can be calculated using Equation 5.2 for a single link:

$$\tau_L = max(T_e, (L_{e_{RIL}}, L_{e_{ESF}} \cdot n_N) + 2J_{L_e}) \tag{5.2}$$

n_N – number of nodes
T_e – event emulation time
L_e – event transmission latency on a *RIL* or *ESF* link
J_{L_e} – event transmission jitter on the link

Equation 5.2: Determination of the maximum event transmission time τ_L.

For an example setup with serial-line-connected *RIL* hardware nodes, $L_{e_{RIL}}$ should become the bottleneck link with the highest latency. Depending on the maximum event data size and the *UART* connection speed (e.g., 115200 *BAUD*[2]), τ_L will range in milliseconds (typically 1-100ms). Contrary, the link latency on a standard host system (e.g., between the simulator and the *ESF*) should not exceed 1 ms. When cascading the time compensation scheme with multiple subsystems, the number of link delays τ_L is added up. For example, considering a standard configuration for the *Split-Protocol-Stack* emulation (*simulator - (a) - ESF - (b) - RIL - (c) - hardware*), there are three independent links *a*, *b*, and *c*.

[2] The most resource-constrained devices are capable of running their *UART* at the standard 115200 *BAUD* rate.

5.3 | Simulator Interfacing and Concept Implementation

As our prototype is based on the *IEEE 802.15.4* protocol standard and the *OMNeT++* simulation environment, we now point out some important details of implementing abstract and generic interfaces among the involved systems and frameworks. We first introduce the basic interface modeling and abstraction, then the implementation of the interfacing as well as our synchronization concept on the specific components.

5.3.1 | Interface Modeling and Abstraction

In order to support the interoperability among the architecture's components there is a strong need for a consistent semantics and data interpretation in all involved subsystems. Within the computer network community, *PCAP* (or *PcapNG* for the *next generation* frame format, commonly and hereinafter simply referred to as *PCAP*) is considered as a de facto standard for network packet readings from a network interface (e.g., *NIC* on general purpose host systems). We have chosen it as control and data exchange protocol frame format among all involved subsystems, not least because it supports the individual extensibility with embedded optional fields and the capability of carrying data frames from multiple network interfaces in one single data stream. We break with the regular usage that *PCAP* streams only carry *DLL PDUs* or *frames*. For the internal message processing and handling of *SDUs*, we have implemented a subset of the *PcapNG IETF Internet-Draft* authored by Tuexen et al. [158]. The following three basic block types are most relevant for our use case:

- *SHB* *(initialization of the PCAP stream)*
- *IDB* *(assignment of the device interface)*
- *EPB* *(processing of the PHY SDUs)*

We have modeled all specific *MAC PDUs* and created respective serializer functions to convert the internal representation and an ordinary byte format into each other. Since there is no specification in the *IEEE* standard of how *PHY-SDUs* should be modeled internally, we have added a simple byte format in which all these protocol primitives for the *data* and *management service* are represented. In Figure 5.3 an example *Physical Layer Service Data Unit (PSDU)* shows an indication of an incoming *DataIndication* (*PPDU*) from a *RIL* hardware node.

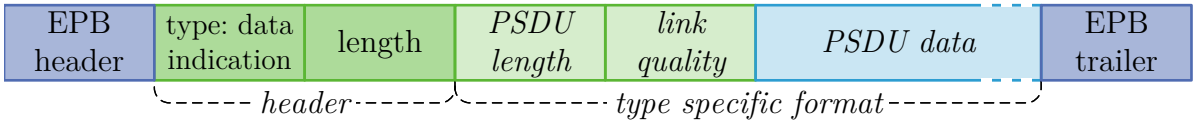


Figure 5.3: Example *IEEE* 802.15.4 *PD DataIndication SDU* byte format, encapsulated in an *EPB*

Since the event transmission latency cannot be neglected, we focus on limiting the *IPC* latency and message jitter to a minimum by applying lightweight stream-based technologies among components of the same host system and networked components. Furthermore, using stream-based *IPC* also keep the sequential order of the event messages. Latency and jitter can be decreased to a minimum using fast stream-based *IPC* via *Unix Domain Socket (UDS)* at the same host system (*localhost*). *UDS* communication is faster than *TCP* communication on the local host system due to the reduced protocol and network driver overhead. In Section 5.4, we show results of reducing the jitter, using *UDS* compared to *TCP* sockets. This increases the performance of the overall system, but does not dissolve the latency issue when communicating between *MAC* and *PHY*.

5.3.2 | Simulation Model Enhancements and External Interfacing

In order to make the simulated nodes capable to connect to real sensor nodes on the emulation hardware platform we have added some emulation features and interface modules to our *OMNeT++ IEEE 802.15.4* simulation model [96], introduced in Section 2.3. In the following we give a detailed overview of our simulation model architecture and its modules. Figure 5.4 depicts the emulation and interface modules as well as the event and information flows from a simulated node to the host system. It serves as basis for the further explanation of our *RIL* simulator implementation.

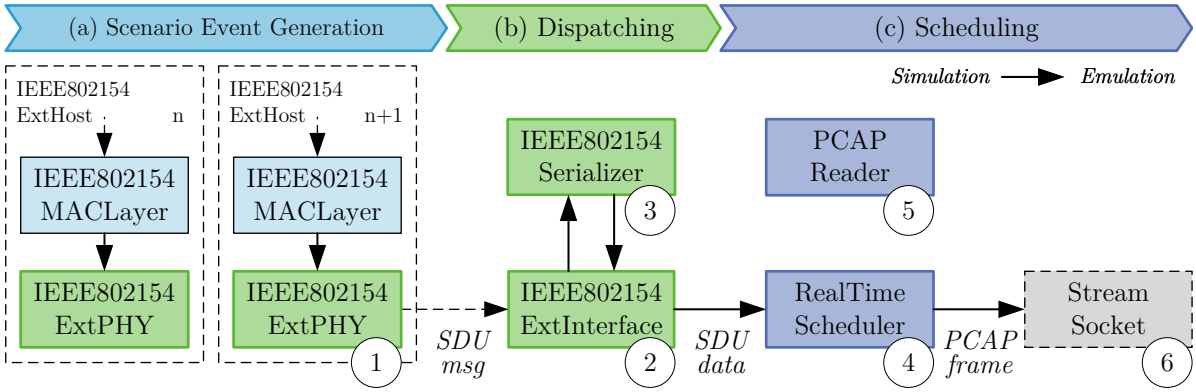


Figure 5.4: IEEE 802.15.4 simulation modules in *OMNeT++*/INET

(a) Scenario Event Generation

To enable the external communication for the *RIL* emulation mode from a virtual wireless node we created the `IEEE802154ExtHost` which acts as representation of a real or emulated wireless node in the simulation system. This module is derived from the standard wireless host of our simulation model [96], but it has no radio interface and is not able to communicate with a virtual/simulated wireless channel. Instead of a *PHY* layer simulation module with gates to a radio interface, we apply an `IEEE802154ExtPHY`

module (ref. ① in Figure 5.4) that models the handling and transfer of all management and data services (*SDUs*) standardized in *IEEE 802.15.4* to the external *PHY*. The `IEEE802154ExtPHY` module can be considered as an abstract *PHY* with no *IEEE 802.15.4*-related but *SAP* interfacing functionality within the simulation. The core function of this module `handleMessage()` deals with events that can be either *MAC*-generated regular simulation events or *RIL* external messages (ref. Listing C.2 for an example *MAC* event handling to instruct external *PHY* to set the *TRX* state).

(b) Dispatching

The communication of the virtual nodes in the simulation scenario is transferred via the `IEEE802154ExtInterface` module (ref. ② in Figure 5.4). The mapping between simulation nodes and *PCAP* interfaces is performed with the help of an interface table that stores the simulation module identifier for the corresponding hardware identifier. The dispatching adds this identification of the node for the later *PCAP* interface assignment and serializes the internal *SDU* message representation of the simulation module to the before mentioned ordinary byte format via the `IEEE802154Serializer` module (ref. ③ in Figure 5.4) which is transferred for processing to the scheduler then. In the case of the *Data Service (PD)* (e.g., a *DataRequest* or *DataIndication*), the encapsulated *MAC* frame is also serialized and included in iteration. In Algorithm 3 an iterative serialization inside the `IEEE802154ExtInterface` module is shown. If the *SDU* message from the *MAC* module is a *packet* it carries an encapsulated *MAC* frame.

Algorithm 3: Serializer call - iterative serialization of a *DataRequest* message

```
Function handleMessageSim(cMessage *msg) ▷ message from virtual MAC
    initialize buffer
    if (msg is packet) then
        Function serialize(msg, buffer)          ▷ iterative serialization
            serialize(encapsulated packet, buffer)
    end
return
```

An incoming *external IEEE 802.15.4 MAC PDU* is deserialized from a *DataIndication* and send via the corresponding `IEEE802154ExtPHY` to the virtual *MAC* simulation module. Hence, the `IEEE802154Serializer` module is responsible for the conversion between the two message representations of *PSDUs* and *MPDUs* (ref. Listing C.1 in appendix Section C.1). For further encapsulated *PDUs*, we also need to call higher layer protocol serializers that are often provided by the higher layer simulation models.

The `IEEE802154ExtInterface` is also responsible for dispatching the *Real-Time-Shift* synchronization events. When an emulated event leaves the simulation, the *virtual* event

scheduling needs to be paused for the *RSV* time interval and resumed afterwords. Because the simulation scheduler needs to be fully reactive for incoming *external* event messages (e.g., *confirmations*), the *pause* and *resume* events are only inserted in the *FES*, but the actual event scheduling is not interrupted. The synchronization event handling is taken over by the *scheduler* module.

(c) Scheduling

The basic concept for connecting an *OMNeT++* simulation with a host system was first published in [66]. We have adapted and extended this approach to enable the *RIL*-related node operations over *PCAP* streams with the *Real-Time-Shift* pseudo-real-time event scheduling. The `RealTimeShiftScheduler` (ref. ④ in Figure 5.4) is derived from *OMNeT++*'s `cRealTimeScheduler` class and enables sending and receiving packets to and from the *external* world via the configurable stream socket (e.g., *TCP* or *UDS*) (ref. ⑤ in Figure 5.4). In a *DES*, the event scheduler is one of the most important components, as it controls the event processing and manipulates the *FES*.

The scheduler class provides a function, called `setInterfaceModule()`, which enables the connection of external interfaces (e.g., a *TCP* socket or *UDS* in our case) to the simulation. The scheduler function `getNextEvent()` is synchronized to the host systems real-time clock and checks the socket periodically (e.g., the scheduler accuracy of the base class in *OMNeT++* is set to 5ms). We implement the pseudo-real-time event scheduling according to the fundamental approach of the *Real-Time-Shift* synchronization in Figure 5.2. Thus, we introduce two new particular core event types (ref. the event scheduling main routine in Listing C.3 and the event shift procedures in Listing C.4 for *OMNeT++*):

- `pause()` *(shift already scheduled virtual events and hold pseudo-real-time clock)*
- `resume()` *(resume pseudo-real-time clock)*

With respect to the *PCAP* file format, the scheduler implements various functions for writing and especially reading the specified blocks to and from the socket stream in the separate `PCAPNGReader` (ref. ⑥ in Figure 5.4) module. For the handling of *PCAP* events, we only implemented the three basic block types that are relevant to our use case. These are:

- `handleSHB()` *(init* PCAP *handling based on the* SHB*)*
- `handleIDB()` *(assign hardware interface module from* IDB*)*
- `handleEPB()` *(process* PHY SDU *event from* EPB*)*

For transmitting *SDUs* from the dispatcher simulation module, the specific *PCAP* block header is generated and the resulting frame is send out. On the return path of the messages, the *PCAP* blocks from the socket byte stream are identified in the receiving process. The scheduler interacts with the `PCAPNGReader`, which assembles the resulting frames from stream segments.

5.3.3 | Radio-in-the-Loop Event Stream Forwarder

The *ESF* is the implementation of the pseudo-real-time scheduling and the intermediate system between the simulation and emulation domain. By default, a simulator is not able to handle the connection to multiple hardware interfaces at the same time because of running in a single thread. Furthermore, a direct communication from the simulator to hardware resources via different interfaces is neither simply to achieve nor a transparent and modular solution. With the transparent *ESF* component in the overall emulation setup, it surrenders a resulting cascaded time event transmission latency compensation scheme. The cascade where the individual node scheduling happens is not visible to the simulation process.

The implementation of the core component, the *event scheduler*, strictly follows the *DES* paradigm and is highly inspired by *OMNeT++*'s message handling within the `cScheduler` class. The implementation uses threads and acts as a dispatcher and aggregator which interchanges stream data in the *PCAP* format. On the hardware side, assigned destinations can be single sensor node platforms, individual transceiver chips, or whole testbed control systems. While *PCAP* 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 5.5 depicts the event processing according an example configuration setup with multiple serial-connected devices.

The *ESF* multi-threaded application processes *PCAP* frames to and from multiple inputs and outputs and enables their manipulation and display. Furthermore, the event process is responsible to schedule the events in accurate real-time. In Listing C.7 in appendix Section C.2, the *ESF* scheduler main thread procedure is shown. Like a *DES*, the event scheduling implements a *FES* which contains all events to be handled in the respective modules. When a new event is received from the simulation, other coupled systems, or device hardware, the corresponding execution timestamp is generated and this event is queued into the *FES*. Once the execution time is reached, the event is forwarded accordingly (ref. line 419 in Section C.2).

This approach allows us to connect between the different evaluation systems. We use in-band signaling for the synchronization based on the *PCAP* frame format. *PCAP* can handle multiple interfaces (*Interface Description Blocks (IDBs)*) with own link layer protocols and options. Hence, our application can aggregate or filter data from different end devices to compose a single data stream for further processing in the simulation system. Our first experiments are based on a *TCP* socket connection to the simulator. Since the simulator and the stream forwarder are running on the same Linux host system in our case, we also implemented and primarily use *UDSs*. The forwarder has a modular and easy extensible structure with several stream-based connection interfaces on Linux systems, e.g.,

- *TCP* *(for standard remote interfacing)*,
- *UDS* *(for interfacing host-based emulation)*,
- *REST* *(for interfacing with high-level APIs)*
- *Serial Port Terminal* *(for serial-line connected device hardware)*,
- *Pseudo Terminal* *(for virtual device hardware)*, and
- *File I/O* *(for tracing and testing)*.

A great advance of this solution is that the simulation system does not need any information about the emulation domain or implement possibly specific interfaces. Figure 5.5 depicts a possible setup as a *Terminal Forwarder* with one simulation interface and multiple connected hardware devices.

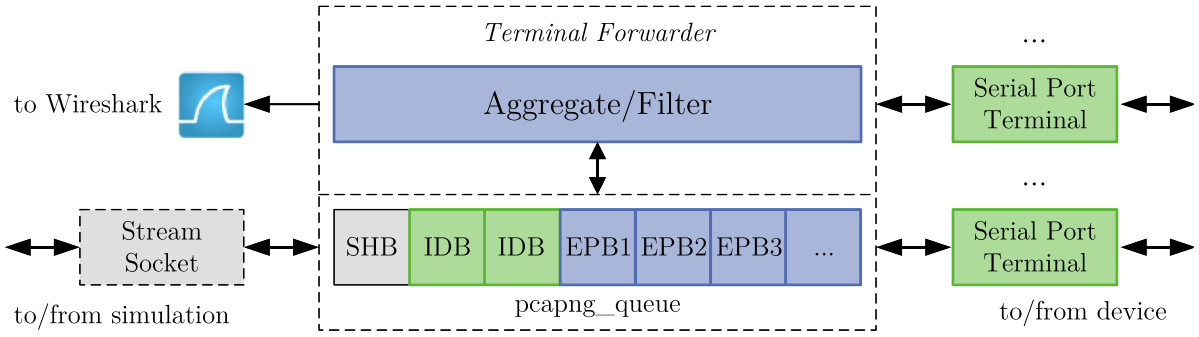


Figure 5.5: Example of a *ESF* setup with multiple serial-connected *RIL* devices

The stream forwarder is not allowed to lose any packets that arrive at the various interfaces at the same time. Therefore, we have implemented every single interface connection and the central forwarder logic as separate threads. As a common data structure for processing the *PCAP* data of different interfaces, we have implemented a thread-safe queue. For example, to consolidate packets from different hardware instances into a single output stream to the simulator we manipulate each packet to ensure the correct interface assignment of each individual *PCAP* frame of this hardware. Vice versa, we have to dispatch all packets from the simulation domain to the respective hardware devices based on the interface identifier.

Furthermore, we have implemented an interface to the de facto standard packet analyzer *Wireshark* (ref. Figure 5.5). Thus, we are able to send the *PCAP* frames at run-time to the Wireshark software application for monitoring and further processing of the transmitted streams. A custom packet dissector visualizes the *IEEE 802.15.4 PHY SDUs* for analyzing and debugging purposes. For our purposes, we use the *PCAP* block types *Section Header Block (SHB)*, *IDB*, and *Enhanced Packet Block (EPB)*. The *SHB* and one *IDB* for every used hardware node are exchanged at the beginning of the scenario execution. The *EPBs* represent the *MAC* data frames.

5.4 | Evaluation and Discussion

With the following evaluation scenarios and measurements, we demonstrate the feasibility of our approach running our *IEEE 802.15.4 OMNeT++* simulation model connected to the *ESF* emulation runtime backend with attached virtual or real hardware *RIL* prototypes. We validate the simulation model implementation by comparing the message sequences with the given charts in the standard specification (cp. *MAC-PHY* interaction message sequence charts in [9, Sec. 7]). Furthermore, we point out several performance measurements of the implemented components and finally conclude with a discussion of the evaluation results.

5.4.1 | Emulation Scenario Model Validation and Verification

As an example reference scenario, we choose the start of a *PAN* in which only a single device tries to start a network as a coordinator (*PAN* start based on an active channel scan [9, pp. 217ff]). The scenario is executed according to a pre-configured settings of the simulation model (cp. [96] and the attached scenario configuration files in Subsection C.1.1 for further details). Figure 5.6 shows the scenario event message log in *OMNeT++* that corresponds to the message sequence charts in the *IEEE 802.15.4* standard specification.

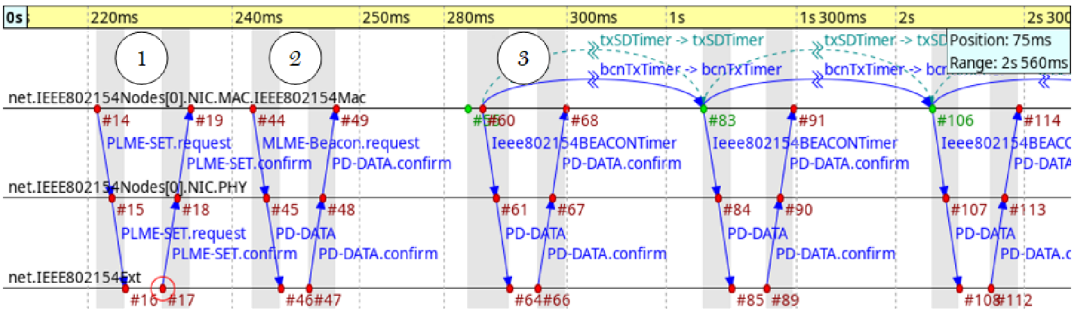


Figure 5.6: *OMNeT++* event message log of the test scenario run. The event log is cleaned to show only the most important modules and events of this simulation scenario for a clear overview (e.g., all *PLME SetTRXState* messages, changing the radio transceiver state, are removed).

At first an active channel scan according to the *IEEE 802.15.4* protocol specification (① in Figure 5.6) is performed. The *PHY* switches to the channel by setting the `phyCurrentChannel` with an *PLME SetRequest* accordingly. Then the radio transmitter is turned on and the *PHY* transfers a *Beacon Request* command frame (② in Figure 5.6). After this the transceiver is put into receive mode and waits for incoming *beacon* packets. Since there is no beacon received during the active channel scan, the *MAC* finishes the channel scan request. Finally the *MAC* starts a new network and selects a node

as coordinator, which periodically sends beacons (③ in Figure 5.6) according to the calculated *BI* (Equation 5.3).

$$BI = aBaseSuperframeDuration * 2^{BO} symbols \tag{5.3}$$

Equation 5.3: Calculation of the *IEEE 802.15.4 BI* (taken from [9])

The *PHY* sets the transceiver state to `TX_ON`, transmits the beacon according to the calculated time interval (`Ieee802154BEACONTimer` → `PD-DATA.request`) and then sets the transceiver state to `RX_ON`. For each beacon, the beacon timer is restarted. As with the scenario in Figure 5.6, the beacon packets are scheduled with the static simulation time difference of 983.232ms, according to the calculated beacon interval with a *Beacon Order (BO)* of 6.

Protocol Message Sequences

The correct flow of the *SDU* event messages can be observed at the *ESF* and the simulation as backend and initiator. In *OMNeT++* (Figure 5.6) the interchanged *SDUs* in the protocol sequences according to [9] can be analyzed in detail. The event sequence at the *ESF* is recorded or can be live viewed with the associated *PCAP* packet dissector for *IEEE 802.15.4 PHY* in *Wireshark* (ref. Figure 5.7). If the *PCAP* events are recorded it is possible to execute or replay a simulation run without coupling the simulator with a realistic analog hardware radio environment.

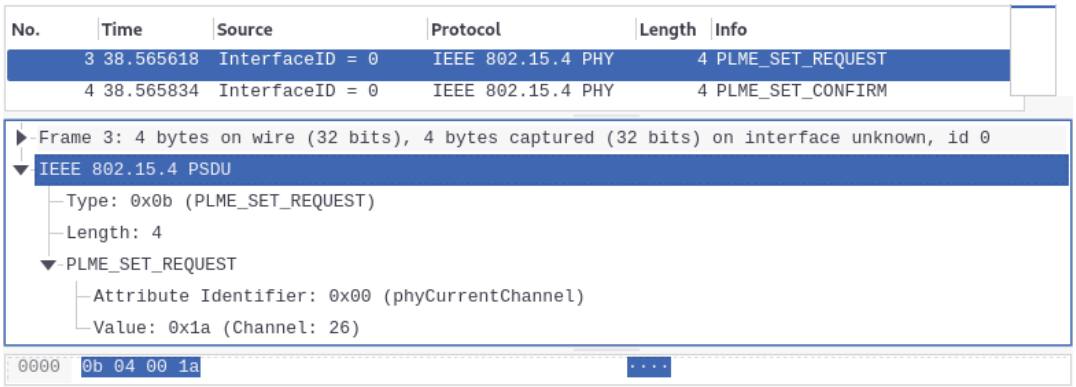


Figure 5.7: *PCAP* trace of the scenario protocol sequences (*IEEE 802.15.4 PHY SDUs*) in *Wireshark*. The *SetRequest* for switching the radio channel (① in Figure 5.6) is highlighted.

The execution of the *SDU* commands and the transmission of the respective real radio packets (*PDUs*) from *RIL* hardware nodes depends on the transceiver implementation and is validated by a radio *Packet Sniffer* reference module. This validation is part of the *RIL PHY* modeling in the following chapter (ref. Figure 6.4 in Section 6.4).

5.4.2 | Event-Stream Forwarder Performance and Accuracy

As our forwarder application acts as a transparent bridge between the simulation and the emulation domain, it must ensure the fast reception, processing, and transmission of *PCAP* events. The performance and accuracy of the *pseudo-real-time* event scheduling and time compensation can be evaluated by performance and accuracy measurements when aggregating, splitting, and filtering the data streams. Figure 5.5 shows the corresponding test setup for the *ESF* performance and throughput measurements considering *File I/O* for *PCAP* event tracing, as well as real transceiver chips (via *Serial Port Terminals*) and full virtual components (via *Pseudo Terminals*) as hardware devices. The *ESF* records the arrival and departure time for each processed *PCAP* event frame by means of precise time-stamping based on high resolution clock timers. For running pure performance and accuracy measurements, we performed the measurements with generated *IEEE 802.15.4 PHY PCAP* traces.

Throughput

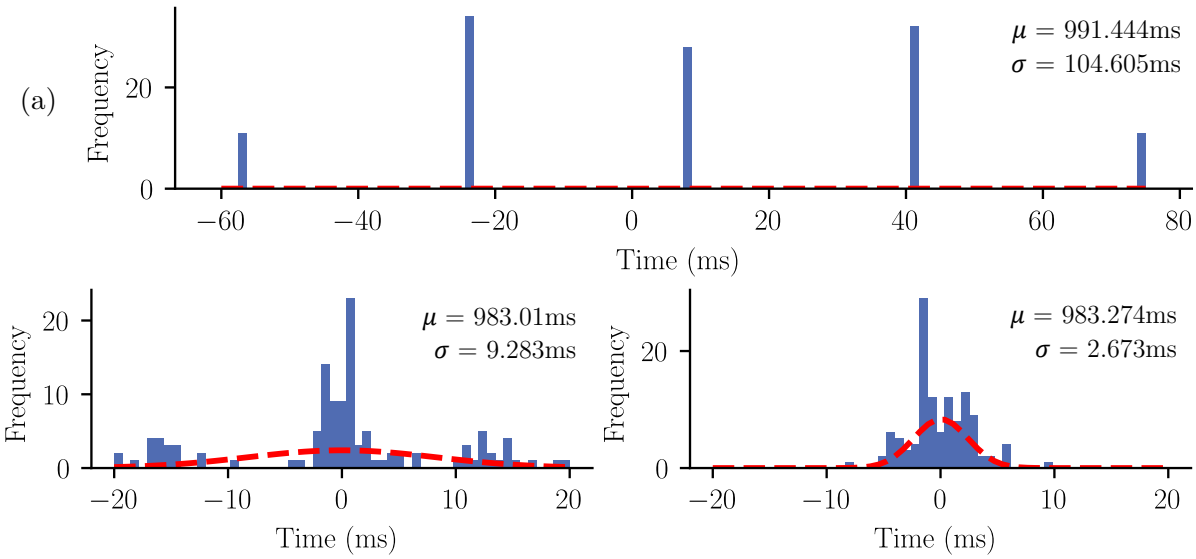
A stress test of the *ESF* event handling is performed by measuring the maximum throughput with generated *PCAP* traces over *TCP* connections. Within the corresponding test setup, a *PCAP* traffic generator sends a data stream from a trace file to the *ESF* which assigns and forwards the analyzed messages to a virtual node. The node simply reply to this messages by repeating the exact same data content[3] which is in return aggregated by the *ESF* and sent back to the generator. For this measurements, we vary the data size of the encapsulated frames which are *IEEE 802.15.4 PD DataRequest*'s in this case. We did not observe any dropped frames; all packets sent by the *PCAP* traffic generator were received by it after they passed through the *ESF* and were returned by the node.

Looking at a data transfer of maximum sized *IEEE 802.15.4* data frames (10,000 data frames with 127 B each – overall *PCAP* data amount of 1.52 MB), we measured an exemplary throughput of 91.5 Mbit s^{-1} from the sender (*TCP* stream traffic generation) to the event stream forwarder and 91.4 Mbit s^{-1} from the sender to the receiver module (*TCP* stream traffic repetition) on an Ubuntu Linux[4]. Since the traffic at the local host gets processed by a loopback adapter in the kernel, we cannot evaluate the precise throughput readings, but our measurements show a sufficient packet throughput for transmissions in larger wireless networks with low data rates (e.g., 250 kbit s^{-1} for *IEEE 802.15.4* O-*QPSK* 2.4 GHz). Thus, we can demonstrate that our packet handling routines play a subordinate role on the overall data throughput and will thus not cause a bottleneck of the overall *Split-Protocol-Stack* emulation.

[3] The virtual nodes are set up as simple *PCAP* repeaters with no *PHY*-related functionality.
[4] Ubuntu Linux (64 bit) virtual machine with 7 vCPUs (Intel Xeon X3470 Quad Core @ 2.93 GHz)

Scheduling and Event-Processing Accuracy

Contrary, when connecting *RIL* nodes in a simulation scenario with timestamped event messages, a high *ESF* scheduling accuracy plays a superordinate role. With a single *UART*-connected *IEEE 802.15.4* chip hardware transceiver, comparative measurements to evaluate the real radio packet transmission jitter were performed based on the aforementioned *OMNeT++* reference simulation scenario as a first step (the 6 *BO* result in 983.232 ms time difference between packets). The Figure 5.8 clearly shows the impact of the *ESF* offset jitter compensation scheme on the accuracy of the received real radio packets compared to pure *TCP*- and *UDS*-based transmissions. This analysis also indicates the more reasonable use of *UDS* instead of *TCP* to send events from the simulator to the *ESF* on a local host system. The effects of the reduced protocol-overhead result in a smaller overall radio packet transmission jitter which eventually will lead to a smaller scenario *pause time* τ_L.



(b) UDS (no compensation) (c) UDS (offset compensation)

Figure 5.8: Histogram of the real-time jitter: **(a)** TCP (no compensation), μ refers to the mean value, σ refers to the standard deviation and beacons are scheduled from the simulation scenario with a size of 51 B and a *BI* of 983.232 ms.

For evaluating the actual scheduling accuracy of the *ESF* in scalable scenarios, parameter studies are performed with a stream-generator application that sends time-accurate *IEEE 802.15.4 PD DataRequest SDUs*. The *RIL* nodes are based on the *native* virtual platform running their event handling routines as application-level processes concurrently on the *Split-Protocol-Stack* emulation host system[5]. The *ESF* dispatches the individual *emulated* events to the corresponding nodes via *Pseudo Terminal* connections and replies

[5] Chip *RIL* transceiver implementation using the *RIOT* operating system on the *native* platform.

the aggregated returning *IEEE 802.15.4 PD DataConfirm SDUs* from the nodes back to the initiator, where the resulting event stream is recorded. All measurements were performed based on the *Real-Time-Shift* scheduling and demonstrate the accuracy of *ESF* with respect to internal parametrization and external input event data streams.

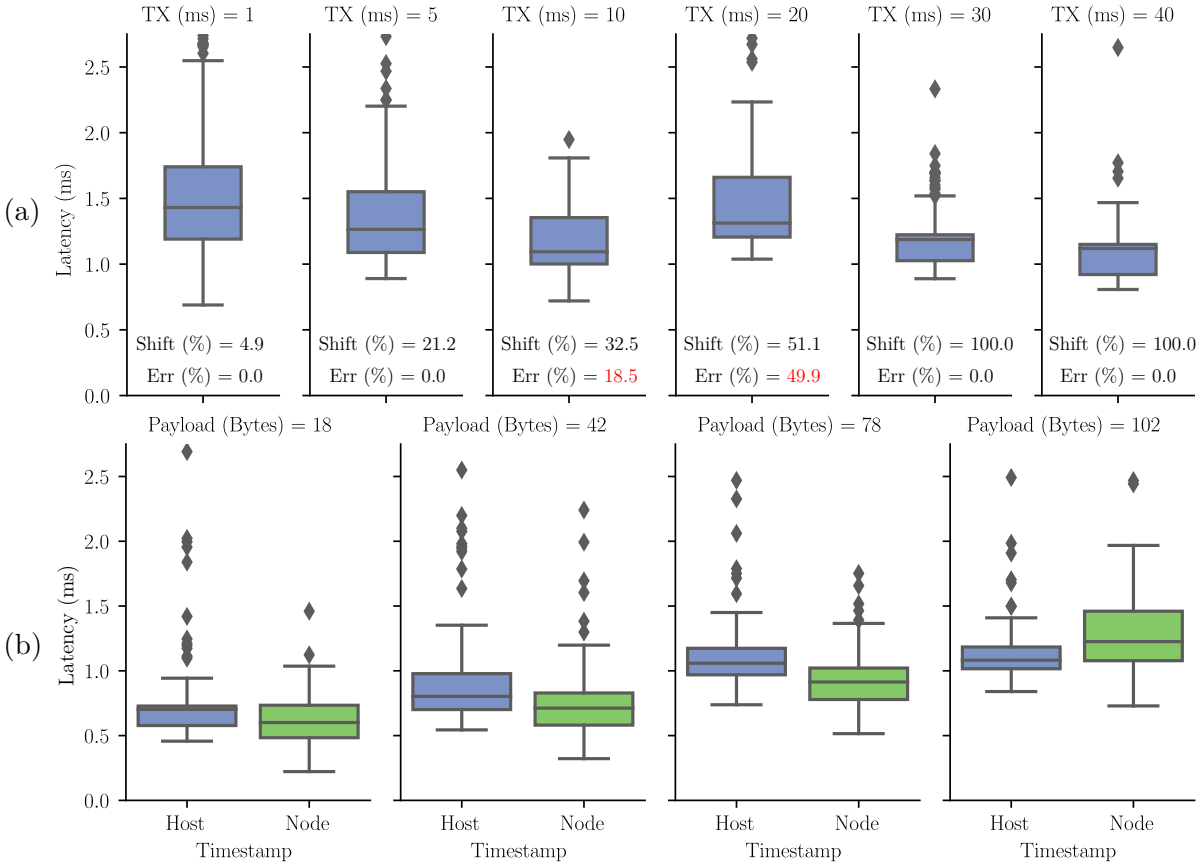


Figure 5.9: The *ESF* scheduling accuracy as a function of the event (*request*) transmission parameters: **(a)** transmit interval and **(b)** payload size. Mean value and standard deviation are reported with parameters: $\tau_L = 10\,\text{ms}, n_N = 20$ (payload size 48 B in **(a)**, transmission interval 50 ms in **(b)**).

The input time interval of the event messages (*requests*) affects the accuracy of the scheduling. Figure 5.9a shows that at an interval below τ_L the standard deviation of the event transmission latency is reduced approximately by a factor of two, compared to send intervals greater than $2\tau_L$. If the interval is exactly once or twice the offset time ($TX(\text{ms}) = [10, 20]$, $\tau_L = 10\,\text{ms}$), several events are sent together with its predecessor (resulting in an event scheduling error rate of up to 50%). However, within a *RSV* (event $Shift < 100\%$) only events with an offset smaller than τ_L can occur, which is why these can be excluded from consideration[6]. The accuracy increases significantly when a separate *Real-Time-Shift* is performed for each *emulated* event ($Shift = 100\%$).

[6] These values were removed from the accuracy consideration in the figure based on this finding.

Figure 5.9b exemplarily demonstrates for the offset compensation time $\tau_L = 10\,\text{ms}$ in a scenario with 20 connected nodes that the payload consequently affects the latency and increases linearly with the packet size due to the serial-port terminal connection to the nodes. The latency mainly depends on the properties of the terminal connection and the event handling routines on the *RIL* hardware. It increases of course when real node hardware with comparatively slow *BAUD* rates and more inaccurate clocks is used. Nevertheless, to evaluate the impact of the event creation strategy different measurements were performed to record the *PCAP* timestamps either on the host system or right on the node hardware. The results clearly indicate an influence on the accuracy is not observable independent of the individual node or timestamp creation (host vs. node) even when applying virtual nodes.

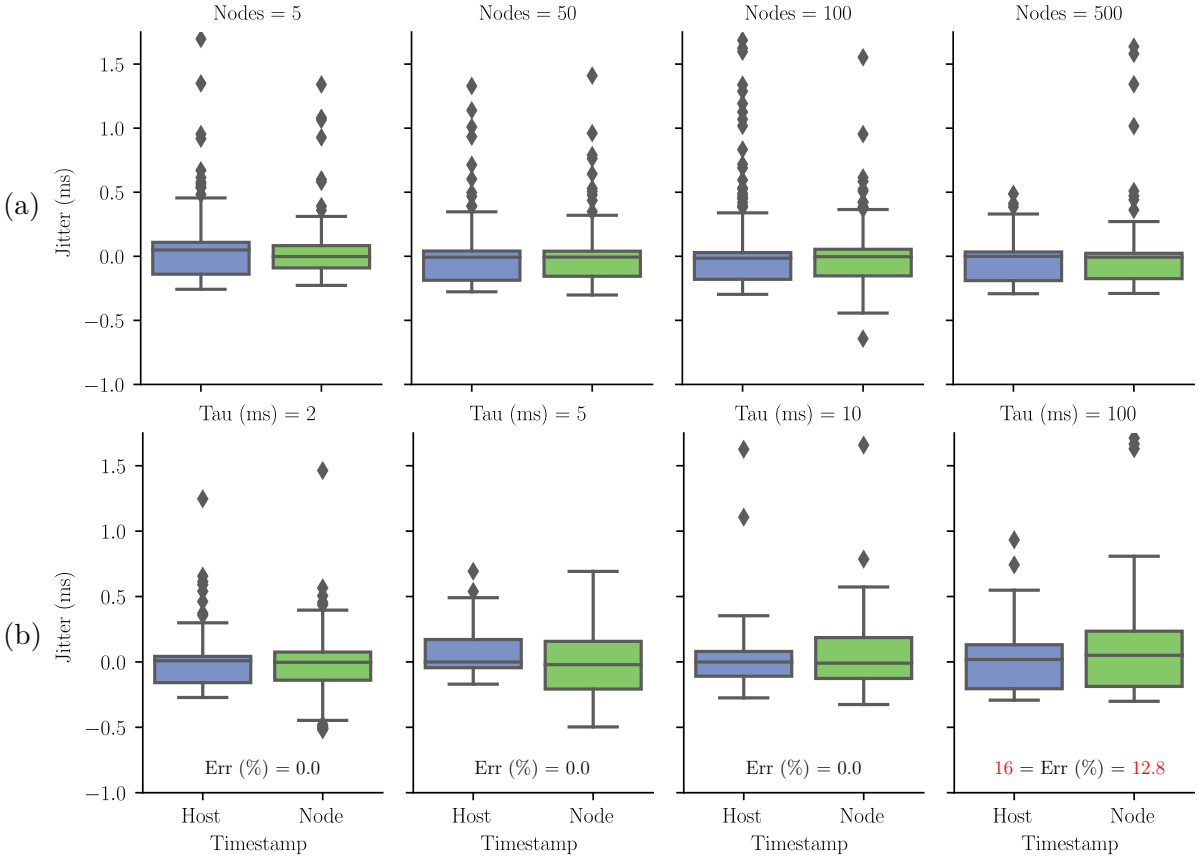


Figure 5.10: The *ESF* scheduling accuracy as a function of internal parameters: **(a)** number of nodes and **(b)** offset time τ_L. Mean value and standard deviation are reported with static input event (*request*) parameters: payload size 48 B, transmission interval 50 ms (offset time $\tau_L = 10\,\text{ms}$ in **(a)**, $n_N = 20$ in **(b)**).

As the number of nodes increases, the number of threads for the connections to the end devices and event handling routines on the host system increases linearly. The evaluation was carried out exemplary for different numbers of nodes up to 500 to increase the process load at the emulation backend (Figure 5.10a). Due to the *Real-Time-Shift* scheduling

with the corresponding high waiting times depending on τ_L, the set of events scheduled in the offset time from the scenario does not increase systematically with the number of nodes. In contrast, the set of events within an *RSV* is highly dependent on the specific application scenario and network topology. Further investigation has shown that even the scenario- and system-dependent offset time τ_L selection has no observable effect on the scheduling accuracy (except for the special cases from the results in Figure 5.9a). At least, no general influence can be derived from our measurements (cf. Figure 5.10b).

5.4.3 | Discussion of the Evaluation Results

We have proven the feasibility of our *Real-Time-Shift* discrete event simulation approach with selected reference simulation scenarios and measurements, and demonstrated its emulation capabilities. The emulation extensions of the *IEEE 802.15.4* simulation model have been validated according to the *IEEE* standard specification to demonstrate the seamless integrability of the *WSN* simulation with external nodes. The protocol primitives and frame structure can easily be extended with future revisions of the protocol standard which happens comparatively often (cp. the overview of multiple amendments a year in the evolution of the *IEEE 802.15.4* specification in [23]). Based on this emulation capabilities of the simulation model, we can verify the standard-conform operation of a specific layer including the *PHY* by recording the inter-layer protocol sequences in simulation scenarios without the need of standard-conform real hardware resources. Thus, a high insight into all link layer protocol operations with a simultaneous interaction with the real radio channel can be achieved. Furthermore, the recorded simulation traces from pure virtual scenarios can be played back in real-node testbeds applying real radio interactions from simulation generated *SDU* traffic. This means that our approach allows a physical radio transceiver to be put into any specified state within the simulation. Thus, incoming responses from the hardware reflect true *PHY* behavior (inter-layer communication). Up to our knowledge, this is a real innovation in the field of hardware-device coupled wireless network emulation, in which only approaches exist so far to transfer single *MAC* frames out of the simulation in the context of inter-system communication.

Using the high-precision microsecond event scheduling, we have shown how *RIL* event execution is initiated by the *ESF* emulation backend. As the results illustrate, using the *pseudo-real-time* approach, a normally hard-to-predict execution of emulated events on hardware is almost limited to the accuracy of the emulation backend host operating system scheduler. For the specific evaluation host system, a mean event-processing accuracy of the *Real-Time-Shift* network simulation can be calculated which depends on the accuracy of the operating system process scheduling (arithmetic mean standard deviation of the *ESF* event scheduling latency $\sigma = 0.35\,\text{ms}$). Due to the evaluation on a general-purpose host system, no generalization, however, can be made from this specific

value. The absolute accuracy depends of course on the operating system used, kernel process scheduling, and crucially on the specific *CPU*.

Furthermore, it can be ensured that the accuracy of the event scheduling does not significantly depend on the input event stream and the configuration parameters of the *ESF*. The strategy for performing the creation of the timestamps on the host system to reduce the complexity of the *RIL* hardware and the associated synchronization does not limit the accuracy. On the contrary, in comparison with initial measurements using real sensor node hardware (Figure 5.8c), it has been demonstrated, that the reduced accuracy is caused by the hardware itself and not by the *ESF* scheduling. Thus, an accuracy in the sub-millisecond range can be achieved independent of the parametrization. On an operating system with a real-time kernel or even a dedicated emulation backend system, the accuracy can be increased further and, above all, the number of outliers can be eliminated sufficiently. Therefore, running the *Real-Time-Shift* network simulation on a real-time system is appropriate for practical use.

What has to be considered in a specific *RIL* emulation scenario is that the maximum amount of concurrent emulated events is limited to the length of the maximum latency constant τ_L. Within the range of τ_L, all emulated events must be transmitted over the connection interface. Depending on the actual throughput of the connection interfaces between the simulator and *ESF* components, the offset time τ_L can reach a high value with regard to a high amount of nodes and events in the scenario. For the evaluation on a single host system, the *PCAP* events are transmitted with an average latency of approx. 1.0 ms. Thus, the overall emulation execution time depends on the number of emulated events per second and the actual choice of the offset time which can completely be different according to the number of nodes in a scenario and the frequency of emulated events. Since in *WSNs* the radio communication between the nodes is reduced to a minimum (*low duty-cycle*), we argue that the *pseudo-real-time* event scheduling can handle large networks in a non-exhaustive runtime. Theoretically, there is no limit for the number of nodes for the *Real-Time-Shift* network simulation. Of course, the runtime cannot eventually be determined across the board without an exact scenario application context.

6 | Radio-in-the-Loop Physical Layer Modeling

The motivation for modeling the *PHY* as *HIL* interface layer arises according to the *Split-Protocol-Stack* methodology. As standard-conform transceiver hardware usually implements a radio access technology as a network driver, incorporating a seamless interaction between the *MAC* and the *PHY*, there is a need for investigating in modeling the *PHY* as high configurable standalone component. Our *RIL* modeling strategy allows for an event-driven channel access via standard-compliant *SDU* message exchange of a given protocol. It eventually enables the generated traffic from a network simulation to be reproduced in a hardware-based analog channel emulation.

This chapter considers the *RIL* approach regarding the modeling strategy and the respective implementation. At first we state our basic strategy for interfacing real node hardware and discrete-event modeling the *PHY* for *RIL* real wireless transmissions. Derived from this strategy, we introduces two different solution approaches with concept and implementation details. Particular attention will be given to the interpretation and discussion of the evaluation results of the two approaches, demonstrating applicability and suitability for the *Split-Protocol-Stack* emulation.

6.1 | Radio-in-the-Loop Modeling Strategy

First we run experiments with "feeding" real sensor node hardware of the *RoSeNet* emulation system with generated *MAC PDUs* from a simple *TCP/IP OMNeT++*/INET example scenario (cp. [58]). We hence transmitted non-compliant protocol data to the destination hardware over the *HIL* interface in this "first step" scenario. *RIL* was later introduced in [55] with our basic methodology for wireless network emulation, combining radio channel emulation with a *HIL* concept. With *RIL*, we assume that the hardware is represented only by the radio transceiver interface of a communication system. Using real transceiver hardware in wireless network simulations allows for an accurate representation of the *PHY* domains (e.g., *symbol* and *waveform* domains, level ② in Figure 4.2). The modeling of *RIL* wireless transmissions fundamentally splits into three parts: the hardware device interfacing, the real-time-aware scheduling of protocol messages (*SDUs*), and the modeling of the *PHY* services and procedures according to the standard specification. We summarize the basic requirements for interfacing and scheduling simulated protocol sequences on real node hardware and motivate a sophisticated *RIL PHY* modeling practice in the following.

6.1.1 | Interfacing and Scheduling Virtual Events on Real Hardware

The key requirements for real-time event processing in the node hardware are *reactivity*, *timeliness*, and *concurrency*. To avoid unpredictable additional event scheduling latency in terms of event processing on the the *RIL* hardware subsystem in the *Split-Protocol-Stack*, this means:

- **Reactivity** is required according to the event detection and generation. Retrieving the inputs from the data links and returning the outputs as soon as the data of the message event is available.
- **Timeliness** is required for an immediate and interrupt-free event processing. Thus, performing the execution of requested *PHY* services must be ensured as fast as possible and without buffering.
- **Concurrency** is needed according to multiple wireless transceivers in the setup. The detection and processing of events must be ensured simultaneously from all nodes at the required simulation time, even if they overlap in execution times.

For *reactivity* and *timeliness*, embedded systems necessarily use precisely tailored node firmware or real-time-capable operating systems and error-resistant data links. For tiny and low-power embedded wireless systems, employing preemptive real-time multi-threading mechanisms can become a huge overhead, which why they often implement lightweight execution models (cp. [159] for a comprehensive overview). For example, the concept of event scheduling in which event handling routines run to completion as in *DES* (ref. Subsection 2.1.3) is also often used in popular tiny operating systems [159], e.g., *Contiki* or *TinyOS*, but it cannot applied for the real-time processing of multiple parallel tasks. Because the operation of the *RIL* transceiver comprises only interrupt-driven tasks but no concurrently running high-level applications, the operating system must not necessarily implement multi-threading mechanisms. Thus, *reactivity* can be achieved with an interrupt-driven *PCAP* event-detection concept on the communication-interface (ref. Section 5.3). *Timeliness* is perfectly assured by running the event-handling routine to completion with an event-based approach or by a high priority, interrupt-free execution in real-time multi-threading.

In addition, to meet the *concurrency* requirement when emulating wireless networks, parallel control and event execution of all nodes can be ensured by two conceivable methods. If the hardware of the node runs completely *synchronous* in time to the simulation system, when an *external* event occurs, the event processing can directly assign the respective time stamp from its local time or delay the execution until the time stamp of the *emulated* event is reached. Initial synchronicity can be achieved by starting the event processing on the hardware execution in parallel. However, due to the effect of the *clock drift*, which becomes more challenging on low-cost wireless devices, it must be ensured that the execution time also remains synchronous over the entire emulation runtime through regular synchronization by means of updates. This can be achieved

using high accuracy time synchronization schemes for *Ethernet*-connected embedded systems, e.g., *PTP* [160], that can achieve an accuracy of less than 100 ns compared to the *NTP* protocol ($>$ 1 ms) that is not sufficient for this purpose. On the other hand, the node can also run completely *asynchronously* to the simulation system, i.e., it cannot perform any time compensation but execute or forward all events *immediately* without paying attention to the timestamps. This requires a high *reactivity* of the hardware system and a constant transmission rate at the communication interface and exactly timed event interaction to and from multiple hardware nodes.

In order to keep the intermediate system flexible in terms of the used end systems we assume an active host time synchronization to decouple the timestamp generation from the intermediate system event scheduling with the *ESF* (c.p. Subsection 5.3.3) but *asynchronously* running transceiver hardware to avoid frequent time synchronizations with the node hardware due to the *clock drift*. Thus, the intermediate scheduling system does not need to take the current hardware configuration, e.g., switches, controllers, link technologies, into account. But, another real-time event scheduler is running independently, closely connected to the actual transceiver hardware. This strategy allows us to neglect the timing on the *RIL* hardware, including the *clock drift*, since the real-time scheduling is done before the event messages are transmitted through a dedicated interface where the transmission latency can be calculated and taken into account beforehand. In case of embedded transceivers, these are often serial interfaces with fixed transmission rates. By means of the event message length, the scheduler is able to start the scheduling in advance of the actual timestamp. With regard to the *ESF* concept, this is a further cascading of the event stream processing, which additionally increases the simulation *pause time* (t_P).

6.1.2 | Physical Layer Modeling

Modeling the *PHY* from a protocol stack's perspective is achieved by modeling the interfaces, protocol primitives, parameters, services, and internal procedures. We focus on the use of *shared libraries* among all *Split-Protocol-Stack* components for the protocol service primitives (*SDUs*), parameter definitions, and data frame formats (*PDUs*) of the given protocol specification. Thus, the *RIL* nodes additionally need to model the *PHY* interfaces, service routines, and internal procedures. Considering the *PHY* modeling basics in Section 2.2, the most important features to enable wireless packet transmission and reception are *modulation*, *carrier sensing*, *packet detection*, and *packet synchronization*. Usually a standard specification does not define how to implement or model these features on specific platforms. Furthermore, modeling of *PHY* functions for wireless transmissions in real hardware can conceptually be achieved in fundamentally different ways, e.g., either in software via the *SDR* concept or by implementing interfaces and firmware for real radio chip hardware. We present these technologically different strategies of modeling and implementing the *RIL* hardware in the subsequent two sections.

6.2 | Chip Radio-in-the-Loop Wireless Transmissions

Event-driven operating systems are the best choice for implementing *RIL* wireless transmissions, as the multitasking is very similar to the event-processing in *DES*. We experimented with implementing our *RIL* concept for the *Contiki* [81] and *RIOT* [84] operating systems. *Contiki* is a very modular tiny operating system for resource-constrained devices and wireless sensor nodes. A process in *Contiki* is defined by an event handler function and an optional poll handler function. Communication between processes (*IPC*) is achieved by posting events.

6.2.1 | Message Event Handling at the Node Interface

To enable the inter-operation with standard-conform transceiver chip procedures we implemented the event-communication with *PCAP* protocol frames via the serial interface. The real-time communication on the serial interface is based on the *polling* mechanism of the kernel which provides high priority to the *PCAP* event communication via the serial interface *UART*. Because no other high-level tasks are running in our transceiver implementation, it causes serial line events immediately be handled by the corresponding routine. We extended this event handling by implementing a *PCAP* event detection and dispatching algorithm, similar to the *RIL* event stream processing of our *ESF* emulation backend. Eventually *PCAP* message events are generated for handling in the main routine of the transceiver application. Our *Contiki*-based transceiver firmware is available at our fork of the official *repository*[1]. We also offer our *PCAP* event handling procedures for the *RIOT* operation system with the driver implementation *uart_pcap*[2].

6.2.2 | Radio-in-the-Loop Real-Time Scheduling

To omit the clock drift problem our chip *RIL* transceiver runs completely *asynchronously* to the *ESF* emulation backend. This means the nodes' transceiver process has no knowledge about the *GVT* of the overall emulation setup and does not care about timestamps in *PCAP* events. However, the accurate event execution is assured by the time-aware transmission from the event *ESF* and the immediate handling of the serial interface. The encapsulated *PHY SDU*s for all *PHY* services are processed by the *PCAP* event handling. Algorithm 4 illustrates the main transceiver process with the corresponding transmit and receive functions in pseudo-code. The required *promiscuous mode* in *line 1* enables a network interface to pass all received network traffic captured from the medium to the system's *CPU*.

[1] *Contiki* (fork of the official project): `https://git.informatik.tu-cottbus.de/boehmse1/contiki/`
[2] *RIOT* (fork of the official project): `https://git.informatik.tu-cottbus.de/boehmse1/RIOT/`

Algorithm 4: Chip RIL Transceiver Process

```
(1)  Precondition : Init radio driver in promiscuous mode
(2)  Function receive()                                  ▷ radio driver callback
(3)  |  generate DATA_indication PHY SDU
(4)  |  serialize and send indication via PCAP serial interface
     return
(5)  while (running) do                                          ▷ main loop
(6)  |  wait until PCAP serial interface event occurs
(7)  |  deserialize encapsulated request PHY SDU
(8)  |  Function handle_Message(&msg)
(9)  |  |  decode SDU and handle PHY request   ▷ radio driver operation
(10) |  |  generate confirmation PHY SDU
(11) |  |  serialize and send confirmation via PCAP serial interface
     |  return
     end
```

6.2.3 | Handling and Dispatching Protocol Primitives

For processing the *requests* from the serial interface, we utilize the *NetStack* concept from *Contiki* which defines four driver layers (*radio*, *duty cycling*, *MAC* and *network* layer) following *OSI* (ref. Figure 1.1a *layer 1-3*). The interaction with the radio chip hardware only needs to utilize the *radio driver* (`NETSTACK_RADIO`) with the radio parameters (`RADIO_PARAM_*`). The data link and network layer related drivers are simply turned off in the transceiver configuration (cp. *null*_driver* in Listing C.9). This makes the transceiver operate like a device driver from the network interface controller's perspective, where simple function calls are mapped to execute the commands from the *MAC request SDUs* (ref. *line 9* in Algorithm 4). Some example driver operations are

- `NETSTACK_RADIO.get_value(RADIO_PARAM_TXPOWER)` (get the *TX* power),
- `NETSTACK_RADIO.channel_clear()` (perform a *CCA*), or
- `NETSTACK_RADIO.send(&data, size)` (send a radio packet from `data`).

For each *request* from the virtual *MAC*, a corresponding *confirmation SDU* is generated (ref. *line 10* in Algorithm 4) that includes the result (e.g., data, a radio parameter value, or status). An incoming data packet also generates an event in the operating systems kernel and is retrieved via the *NetStack* with its parameters, such as the associated received signal strength, and put into an *indication SDU* (ref. the *receive()* proceedure at *line 2* in Algorithm 4). Thus, the *PHY* modeling is more or less given by the radio driver of the operating system and depends on the features of the used radio chip hardware for the chip *RIL* solution. Excerpts of the of the transceiver main event handling and driver configuration can be found in the Listings C.8, C.9, and C.10 the appendix Section C.2.

6.3 | Software Radio-in-the-Loop Wireless Transmissions

The *SDR* communication system implements traditional *RF* hardware components in software and thus serves as an enabling technology of the cognitive radio, as it is able to change *PHY* parameters and components in real-time. In Figure 6.1, a block diagram of a *PHY SDR* transmitter with O-*QPSK* modulation is abstracted as example. With reference to the general modeling scheme presented in Figure 2.6, it depictes how the *PHY* packet data (*PPDU*) is transformed in software from the *bit* to the *symbol* and *waveform* domain eventually accessing the radio channel via real hardware.

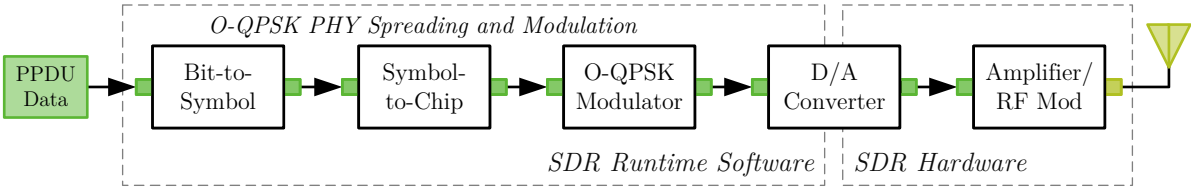
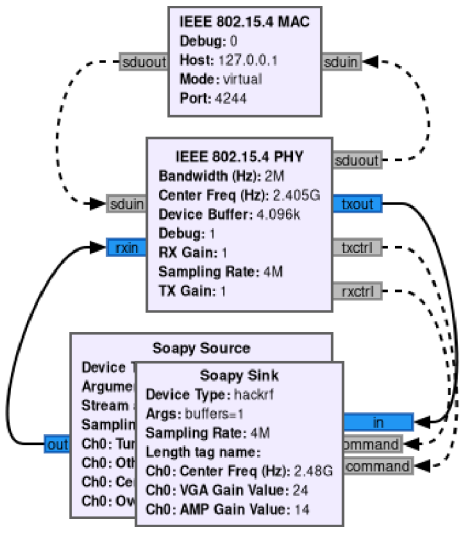


Figure 6.1: Block diagram of an O-*QPSK PHY SDR* transmitter

The *SDR*-based evaluation methodology for the *Split-Protocol-Stack* was introduced in [56]. The implementation of the example IEEE 802.15.4 *SDR* model for the transceiver is based on *GNU Radio* framework, an open source, multi-threaded streaming system in which models are build from basic building blocks. Figure 6.2 depicts the layered software radio transceiver processing flow based on the top-level *SDR* components. We have modeled the transceiver according to the division of the link layer into *MAC* and *PHY* which can clearly be observed in the hierarchical module structure of our model:



- `IEEE 802.15.4 MAC`: The *virtual* mode of the *MAC* module connects to a *PCAP* event-generating host and schedules the encapsulated *SDUs* according to the timestamp.
- `IEEE 802.15.4 PHY`: The *PHY* model component implements the *SDU* message handling, parameter setup, and all specification-related responsibilities, e.g., modulation, carrier sensing, or packet detection.
- `Soapy Source/Sink`: The *Soapy* device abstraction allows the transmitter to use different *SDR* hardware sources, e.g., *HackRF*.

Figure 6.2: The *IEEE 802.15.4 SDR* transceiver model in *GNU Radio*

6.3.1 | Data Link Layer Interface Modeling

We have implemented a virtual *MAC GNU Radio Companion (GRC)* block that provides a *PCAP connection* as the interface to the emulation backend. The *MAC* possesses a combined data and management *SAP* to exchange the *SDUs* with the *PHY*. According to the *GNU Radio* specifics for message passing, the module block `gate` splits into source and sink handling *request* (`sduin`) and *confirmation* (`sduout`) *SDUs*. Besides the protocol serializer for *SDU* $\leftrightarrow$ *PCAP* en- and decapsulation, the *PCAP scheduler* is the core for the real-time-aware event scheduling. This block connects via a stream socket to our forwarder emulation backend and the simulation engine.

6.3.2 | Time-Aware Event Scheduling in Streaming Systems

The *SDR* backend host system, e.g, *GNU Radio*, is a streaming system by default whose components continuously process input to output signals. Turning this into a event-processing system is one of the achievements by our *SDR* modeling concept and transceiver implementation. The real-time *PCAP* event scheduling on *SDR* runtime level is needed to transform the input stream into block events and to compensate the packet jitter between the software radio and the emulation backend first. Since in *GNU Radio* a separate thread is running for each processing block within our scheduling block, we define a microsecond-accurate *PCAP* scheduler running in an additional separate thread. This scheduler is responsible for processing *PCAP* events according to their timestamps from a custom-defined *event queue* in real-time. The regular block thread is forced to receive the *PCAP* stream at the module input to create block events and to queue them accordingly in this event queue. Since the *PCAP* event data already arrives in correct order due to an error-free stream transmission, no sorting has to be performed.

Stream-based inter-process communication naturally produces latency and jitter at the link between the simulation system and the *SDR* execution environment. Running the two runtime systems, i.e., *OMNeT++* for simulation and *GNU Radio* for radio channel emulation on one host, the system can scale down each other. Nevertheless, latency and jitter must remain within fixed limits, so that they can accordingly be compensated for the hardware transmission. The event processing module does this in its core.

According to the latency and jitter range among the involved evaluation runtime systems, a sufficient clock time offset τ needs to be determined. If the whole setup runs on a single host then the local time t_{local} of the two systems is based on the host systems high resolution clock. Therefore, we can neglect the clock drifts. On a multiple host system setup, in contrast, it must be ensured that all host systems are periodically synchronized within the *LAN* via the *PTP*. The initialization of the emulation time is triggered by the first *PCAP* event e in the event queue. According to the delayed arrival of an event caused by latency, the aforementioned offset τ is added to the event time t_e (line 11 in

Algorithm 5: SDR PCAP Scheduler Event Process

```
(1)  Precondition : Init scheduler with fixed time offset τ
(2)  Function processEvent(&time t, &event e)                    ▷ mutex locked
(3)      get time with start offset(t)
(4)      microseconds offset θ(t_e) = t_e − t
(5)      t_start = t_local.now
(6)      while (t_local.now() < t_start + θ(t_e)) do              ▷ active wait
         end
(7)      send message on the output port
     return
(8)  while (running) do                                          ▷ main loop
(9)      fetch first event e from event queue
(10)     set event time t_e with fixed time offset τ
(11)     get time with start offset(t)
(12)     wait until t_e                  ▷ sleep in chunks of 100ms
(13)     processEvent(t, e)
(14)     delete e
     end
```

Algorithm 5). We wait until the execution time is almost reached (<200ms) by putting the scheduler thread into sleep mode first (line 12 in Algorithm 5). Then we calculate the remaining time, wait in a *mutex* locked active loop, and send the message to the block's output port (`processEvent()` in Algorithm 5). Due to reusability considerations of these blocks, the deserialization of the encapsulated *SDUs* and the associated execution of the *PHY* instructions will follow after the event scheduling in the connected blocks. This real-time-aware *PCAP* scheduler is available at our *GNU Radio* module *gr-pcap*[3].

6.3.3 | Software-Defined Radio Physical Layer Modeling

Since the prototype of our practical evaluation is based on the *IEEE 802.15.4* protocol standard, the input of standard-conform protocol sequences is ensured by our accurate simulation model (cp. Subsection 2.3.2 and [97]). Regarding the *SDR* transceiver (see Figure 6.3), we have modeled the *PHY* components and interfaces according to the *IEEE 802.15.4* standard with both *SAPs* to the *MAC* (*PD-SAP* and *PLME-SAP*) and the interface to the antenna (*RF-SAP*). Since we have built our prototype on the O-*QPSK* 2.4 GHz *PHY* currently, the modulation is taken from the *IEEE 802.15.4* O-*QPSK* model in [161]. The implementation can be found on our repository[4]. In order to comply to the

[3] *gr-pcap*: `https://git.informatik.tu-cottbus.de/boehmse1/gr-pcap/`
[4] *gr-ieee802-15-4*: `https://git.informatik.tu-cottbus.de/boehmse1/gr-ieee802-15-4/`

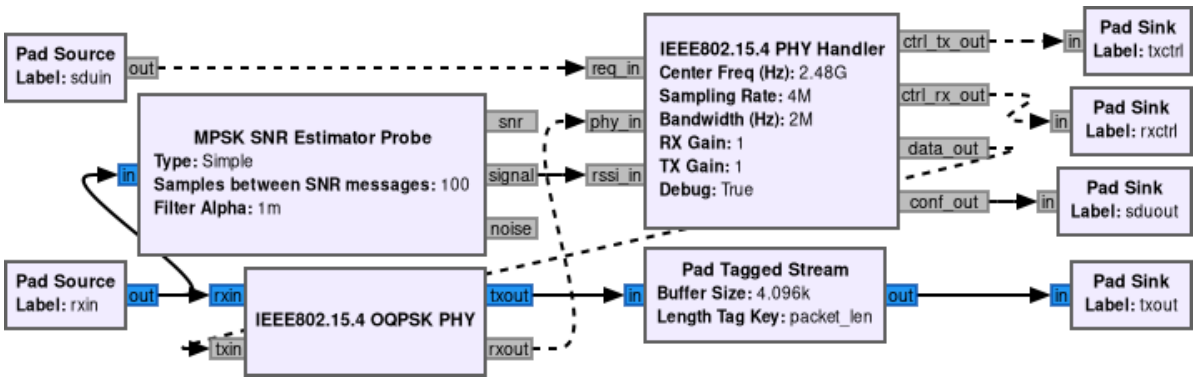


Figure 6.3: *PHY* model of the *SDR* transceiver in *GNU Radio*.

formal protocol standard interface definitions we have modeled individual block gates, physical layer constants, attributes, and specifications for the data and management services at the corresponding *SAPs*. The *message handlers* at the respective gates are responsible to process the individual requests. The interface for the exchange of service primitives are modeled straightforward with respect to the *MAC*. The *RF-SAP*, on the other hand, splits into several gates (see Figure 6.3, not specified in the *IEEE 802.15.4* protocol standard), namely:

- the `rssi_in` to indicate the received signal strength for performing the *ED* and *CCA* according to the specification,
- the `txctrl` and `rxctrl` to configure parameters for the *RX*/*TX* hardware, and
- the `data_out` and `phy_in` to send and receive data via the modulation scheme.

The *IEEE 802.15.4 PHY* is extended with some parameters according to the *RF* interface to the *GNU Radio* runtime environment and *SDR* hardware. Furthermore, we keep a copy of the *GNU Radio* transceiver bandwidth and sample rate settings. The additional parameters are needed for the calculation on the *RF* interface, but do not impact the protocol procedures. While some parameters depend on the specific modulation scheme and frequency which is initialized from the corresponding simulation module, several other parameters depend on the actual settings of the *SDR* periphery and are set by calculating and assigning the appropriate values, e.g.,:

- *phyCurrentChannel* *(RF channel for RX & TX),*
- *phyCenterFrequency* *(calculated from current channel),*
- *phyBandwidth* *(single channel bandwidth),*
- *phyTransmitPower* *(transmission power relative to* 1 mW*),*
- *phyTXgain* *(calculated from transmission power),*
- *phyRXgain* *(receiver sensitivity),*
- *phySignalStrength* *(calculated from* SNR *probing).*

Energy Detection

For the energy detection on the radio channel, we add a *GNU Radio Phase-Shift Keying (PSK)* probe module to the receiver to monitor and retrieve estimations of the signal strength. Based on the probes, we calculate the energy detection result as a signal strength value (8-bit unsigned integer according to the standard). We normalize the range of the measured values with a linear function based on empirical examinations (ref. Equation 6.1). The *signal* corresponds to the measured value from the module and *signalStrength* to the normalized value. The lower and upper limits are set based on the given transceiver and the standard specifications (minimum receiver sensitivity: -85 dBm, typical nominal output power: 1-4 dBm). Thus, the normalized signal strength, for instance, ranges from 0 (-85 dBm) to 255 (1.7 dBm) for our prototype. Based on the energy detection and a fixed threshold value, the *CCA* can be performed as specified.

$$signalStrength = (2.94 * signal) + 250 \tag{6.1}$$

Equation 6.1: Prototype-specific normalization of the received signal strength

Frequency Bands and Modulation

The implementation of the *modulation* is taken from [161]. This module adds the preamble sequence for the *packet synchronization* and performs the coding and modulation (transformation from the *bit* to the *symbol domain*, ② in Figure 4.2) according to the 2.4 GHz O-*QPSK* specifications. Additional *PHY* modulation schemes, algorithms, and experiments are also available (e.g., the 2.4 GHz *Direct Sequence Spread Spectrum (DSSS) PHY* [162], 868/915 MHz *BPSK PHY* [163]), but have not been included in our prototype, yet.

Hardware Interface Abstraction

In order to enable *SDR* hardware diversity we abstract the interface control from specific hardware components by using a *GNU Radio* wrapper for the *SoapySDR* library[5], called *gr-soapy*. This simplifies the parameter setup on device-specific interfaces to configure the bandwidth, the center frequency, and the radio channels. Gain settings of the *SDR* hardware (*RX* and *TX* gain), however, must be evaluated carefully because appropriate values highly depend on the device characteristics and directly influence the calculated *RSSI* and *LQI* values. Individual calculations in the message handler module translates the aforementioned *PHY* parameters requested by the *SDUs* into the corresponding *Soapy* device configuration parameters.

[5] *SoapySDR project*: `https://github.com/pothosware/SoapySDR/`

6.4 | Evaluation and Discussion

At the end of this chapter, we demonstrate the feasibility and performance of our *RIL* modeling strategy and show what results can be achieved when scheduling virtual events on real hardware. We evaluate our *PHY* implementation based on parameter studies and performance evaluation scenarios for radio packet transmissions regarding the *IEEE 802.15.4* standard specification documents [9, Sec. 7]. To generate inbound traffic for the *RIL* nodes we apply the timed event scheduling on the *ESF* with either generated, recorded, or virtual live simulation *PCAP* events. Furthermore, we used the CC2531[6] transceiver module with the *SmartRF Packet Sniffer* software for frame reception. Via the recording by means of the sniffer, we can validate the correct packet transmission as well as some physical parameters and functions of the transceiver. We discuss the application areas applying the two strategies with the *Split-Protocol-Stack* emulation approach and conclude with a short overview and comparison.

6.4.1 | Feasibility and Performance of the Chip Radio Transceiver

For the basic verification of our chip *RIL* model prototype, we utilized the single node *WPAN* simulation scenario (ref. Section 5.4) and experimented with different, common wireless transceiver chip hardware (from *Texas Instruments* and *Microchip*) as well as the *Contiki* and *RIOT* sensor node operating systems. The results are based on the *Contiki* transceiver implementation running on the ATmega128RFA1[7] target chip hardware. Figure 6.4 shows the correct radio packet sequence of the basic single node simulation scenario on the specified radio channel. The correct execution of the radio parameter setting based on the *PHY SDU* events is not explained in detail here, as this is almost exclusively based on the used chip hardware radio driver and functions already included in *Contiki*. The correct mapping of most parameters and functions, however, can implicitly be proven from transmission parameters of a received packet sequence.

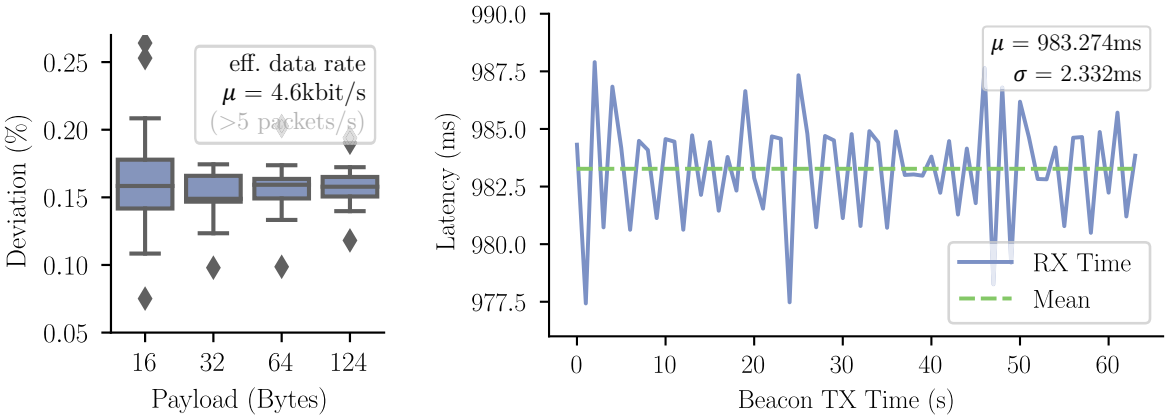
P.nbr.	Time (us)	Frame control field					Sequence number	Dest. PAN	Source PAN	Superframe specification						RSSI (dBm)
		Type	Sec	Pnd	Ack.req	PAN_compr				BO	SO	F.CAP	BLE	Coord	Assoc	
RX 1	+0 =0	CMD	0	0	0	0	0x00	0xFFFF	Beacon request							-31
RX 2	+274024 =274024	BCN	0	0	0	0	0x7F	0xFFFF	0x0007	06	06	15	0	1	1	-31
RX 3	+968721 =1242745	BCN	0	0	0	0	0x80	0xFFFF	0x0007	06	06	15	0	1	1	-31

Figure 6.4: Received radio packets of the chip *RIL* solution with *SmartRF* using Texas Instruments CC2531 sniffer device hardware.

[6] TI CC2531 *IEEE 802.15.4* wireless controller: `https://www.ti.com/product/CC2531`
[7] ATmega128RFA1: `https://www.microchip.com/ATmega128RFA1`

RF Packet Transmission Throughput and Accuracy

At first a stress test was conducted to evaluate the maximum over-the-air transmission rates and the impact of *PCAP* event scheduling on resource-constrained sensor node hardware. The maximum data rate is significantly limited by the *UART* serial interface settings which can lead to non-tolerable error rates when the step rates (*BAUD*) are too high due to the low accuracy of the used oscillator and a possibly unfavorably selected microcontroller operating frequency. In our setup, error-free transmissions are achieved by operating the *UART* at 57600 *BAUD* which results to effective data rates smaller than the over-the-air transmission rate of the *IEEE 802.15.4* 2.4GHz O-*QPSK PHY* ($250\,\mathrm{kbit\,s^{-1}}$) for multiple successive packets. However, the speed of the data transmission is not a major issue, since wireless sensor networks usually do not achieve high packet rates (typically the packet rate in *Beacon-enabled WPANs* is below one packet/s per node). Figure 6.5a depicts the results of the frame processing as deviation from the theoretical limitation due to the *UART* settings. We achieve an overall throughput (independent of the frame size) of nearly 100 % of the theoretical maximum packet rate at the serial interface. Moreover, the packets are transmitted with a very low variation in latency (mean standard deviation over all payload sizes $\sigma = 0.004ms$), suggesting that the *PCAP* serial input handling routines, the frame type classification and frame buffering on the embedded radio chip do not introduce a significant performance degradation.



(a) *TX* packet rate deviation from theoretical maximum.

(b) Radio packet *RX* accuracy over a *Beacon* transmission period of 60s.

Figure 6.5: Chip *RIL* packet transmission performance: *Beacons* are scheduled with **(a)** different sizes, $BI = 0$ms, **(b)** size of 51 bytes, $BI = 983.232$ms.

A limiting factor is, however, the sporadic use of the serial connection in the scenario of Figure 6.5b which is comparatively inaccurate. From the measured values, it is obvious that the fluctuations around the relatively accurate mean value are too high. To compensate this, access to the serial interface from the host system side must be in real-time without device driver-internal *sleeps* or scheduling-dependent waiting times.

6.4.2 | Feasibility and Performance of the Software Radio Transceiver

Regarding the Software *RIL* prototype, we experimented with the open-source hardware platform *HackRF* [8] and used well-established device hardware, drivers, and tools for sending and receiving *IEEE 802.15.4* radio packets and standard laptop hardware[9] with *GNU Radio* for the feasibility evaluation and performance measurements of our *SDR* solution.

Parameter Setting and Interface Control

Contrary to the implementation with standard-compliant radio chip hardware, we performed several feasibility tests and parameter studies by setting the *PHY* attributes to control the transceiver interface. To facilitate the automated parameter setting the inbound *PCAP* stream is based on generated events (`PLME_SET_REQUEST`'s) from the *ESF* emulation backend. Selected results of the parameter measurements are presented subsequently.

P.nbr.	Time (us)	Frame control field					Sequence number	Dest. PAN	Source PAN	Superframe specification						RSSI (dBm)
		Type	Sec	Pnd	Ack.req	PAN_compr				BO	SO	F.CAP	BLE	Coord	Assoc	
RX 1	+0 =0	BCN	0	0	0	0	0x7F	0xFFFF	0x0007	06	06	15	0	1	1	-66
RX 2	+196610 =196610	BCN	0	0	0	0	0x80	0xFFFF	0x0007	06	06	15	0	1	1	-65
RX 3	+196610 =393220	BCN	0	0	0	0	0x81	0xFFFF	0x0007	06	06	15	0	1	1	-64

Figure 6.6: Received radio packets of the software *RIL* solution with *SmartRF* using Texas Instruments CC2531 sniffer device hardware.

`phyTransmitPower`: The *HackRF* device hardware transmission gain range is specified from 0 to 61, whereby a step equals to 1 dB. As the *PCAP* stream contains alternating *SDUs* to set the transmission signal strength (`PLME_SET_REQUEST.phyTransmitPower`) and simple *IEEE 802.15.4 Beacon* packets (`PD_DATA_REQUEST`), we increased the transmission power by exactly one dB for each individual packet with this setup. We performed this measurements with several repetitions using a very small distance of 10 cm between transmitter and receiver, and set the beacon interval to 200ms to speed up the measurements. Figure 6.6 shows the correct radio packet (*Beacon*) sequence of the single node scenario on the specified radio channel. Furthermore, as depicted in Figure 6.7, the transmission power can be adjusted very precisely for wireless transmissions. The received signal strength was in the range from -67 dBm to -13 dBm, with an average error in the absolute uncalibrated *RSSI* accuracy of our receiver device CC2531 ($\pm$4 dB).

[8] HackRF One SDR peripheral: `https://greatscottgadgets.com/hackrf/one/`
[9] Manjaro Linux (64 bit) with 4 vCPUs (Intel Core i7-5600U @ 2.60 GHz).

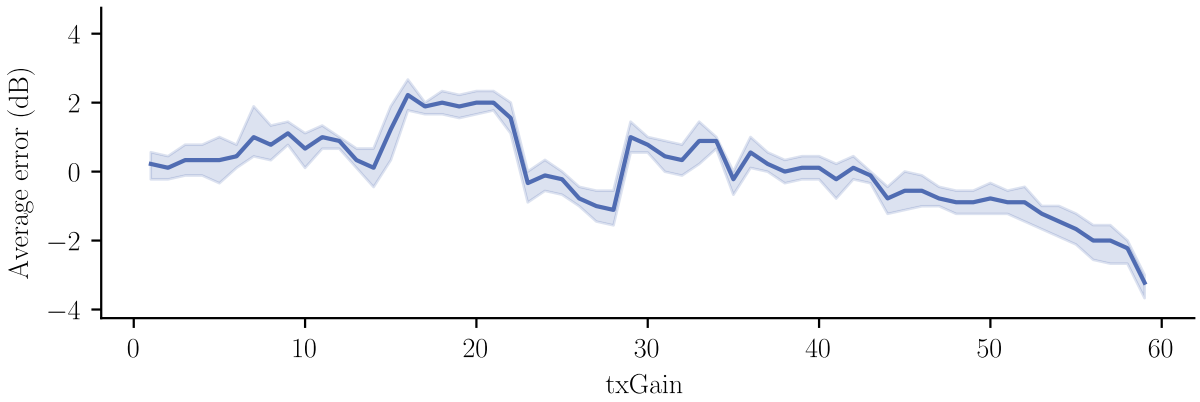


Figure 6.7: Deviation of the mean received signal strength (*RSSI* in dBm with *SmartRF*) as a function of the software *RIL* transmission power.

`phyCurrentChannel`: Based on our scenario, we generated a *PCAP* stream with a decreased *BI* to prove the feasibility of switching the channel for every burst of transmitted beacons over the whole range of the 2.4 GHz channels of the *IEEE 802.15.4* O-*QPSK PHY*. Figure 6.8 depicts the channel hop from channel 15 to 16. According to the limited bandwidth of our *HackRF SDR* receiver, we can only show the frequency range of 2 out of 16 channels in a single waterfall plot from the spectrum analyzer.

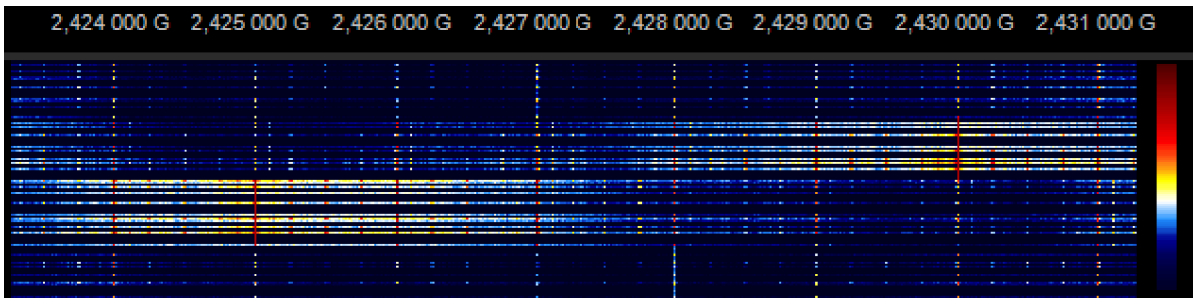


Figure 6.8: Waterfall plot of the channel hop from 15 (2.425 GHz) to 16 (2.43 GHz)

Energy Detection and Packet Reception

A complete parameter study is required to determine favorable device parameters for general purpose *SDR* hardware. Exemplarily, we performed several reference measurements with ordinary indoor channel characteristics in our office to evaluate the packet reception and parameter setting for the *CCA* based on the *ED* signal probing with the specified normalization scheme. This simultaneously demonstrates the feasibility of receiving radio packets. We set up a standard-compliant chip hardware (TI CC2420) placed at a distance of 0.75 m from our software radio module to send a burst of 30 *IEEE 802.15.4*-compliant radio packets each. The Figure 6.9 shows the dependence of packet reception on the *RX* interface gain setting based on the packet rate and signal strength.

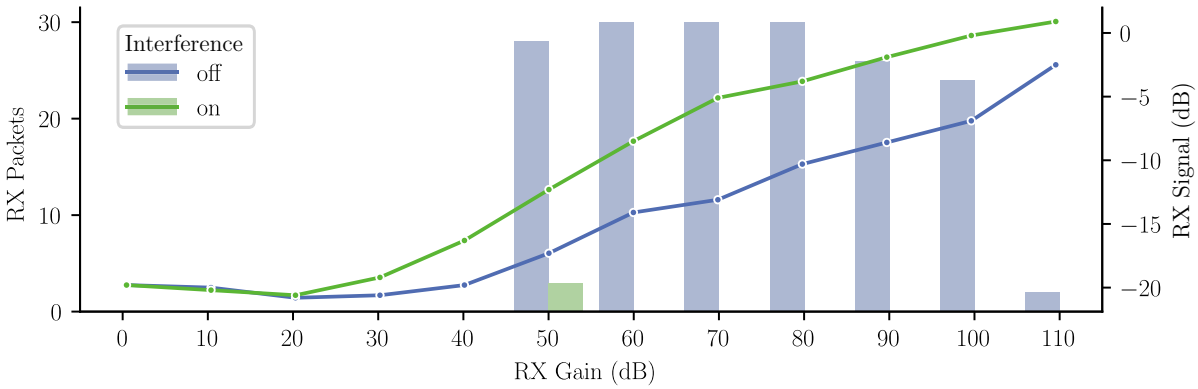


Figure 6.9: Received signal strength and packet count at the software *RIL* model as a function of the receiver sensitivity (*RX* gain)

In the reception gain range of 60-80 dBm, all packets could be received by our software radio transceiver under realistic uncontrollable but largely interference-free environmental conditions. Moreover, a higher reception gain at the antenna has a clearly negative effect on the packet reception (ref. packet reception rate @ 110 dB in Figure 6.9). We assume that the receiver is already slightly overdriven due to the activated baseband amplifier. If the jammer is enabled (*Gaussian* noise with additional signal intensity around the *TX* power) only a few packets are received at all due to the difficult or impossible decoding. Below 50 dB *RX* gain no packets could be received by the receiver because the *SNR* for this environment is not sufficient. Based on such measurements, the threshold value for the *ED* can be determined. Since the *RX* signal strength level is above the threshold in the two cases, a *CCA* triggers a corresponding negative confirmation to forward to the emulation backend (*CCA* mode 1 in [9]).

Event-Scheduling Accuracy

The time accuracy of the transmitted radio packets via the *SDR* hardware decisively depends on the accuracy of the *RIL* scheduler. Figure 6.10 shows the histogram plot of the *GRC PCAP* scheduling jitter for the same scenario as introduced alongside the *IEEE 802.15.4* emulation model validation in Section 5.4. *Beacons* in this scenario are scheduled with a constant overall frame size of 51 Bytes. The normal distribution of the *PCAP* scheduling time is exactly around the *BI* of 983.232 ms according to our model-based emulation scenario setup, but the standard deviation is 330 us. According to the compliance to the time intervals between packets (*IFS*) in cooperative networks, this jitter value is too high. For example, in *IEEE 802.15.4* 2.4 GHz O-*QPSK PHY* [9, p. 30] (symbol period 16 µs), the standard deviation is approx. 20 symbols, while the minimum *Long Inter Frame Spacing (LIFS)* and *Short Inter Frame Spacing (SIFS)* are specified to 40 and 12 symbols. This means in the worst case that the *LIFS* and *SIFS* could slightly overlap due to too long or too short waiting periods.

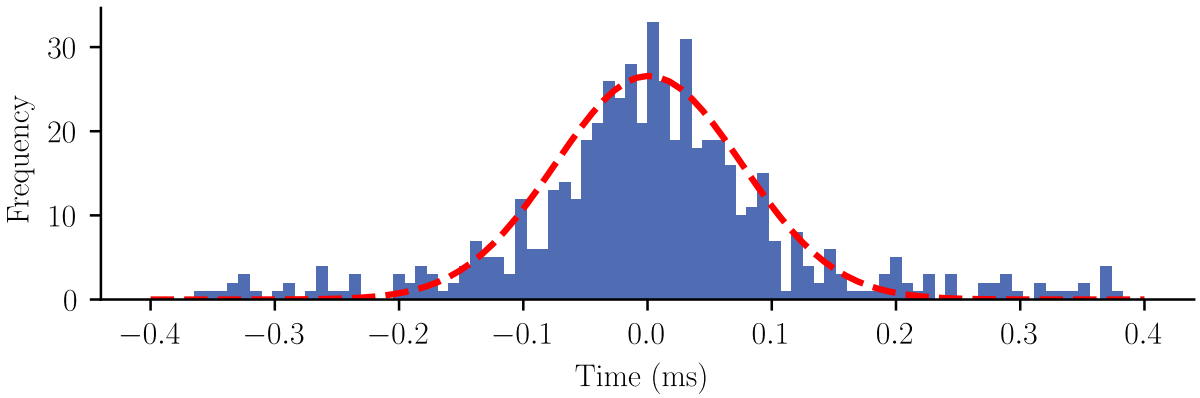


Figure 6.10: Histogram of the *GRC PCAP* scheduling jitter: $\mu = 983.232ms, \sigma = 0.33ms$. *Beacons* are scheduled with a size of 51 bytes and a *BI* of 983.232ms.

The dependence of the frame size is illustrated in Figure 6.11 in which the mean and the standard deviation values of the scheduling jitter are represented as function of the *Beacon* frame size. It can be concluded that no general degradation of the scheduler accuracy can be derived with increasing packet size. On the contrary, the standard deviation σ decreases for smaller and larger frames. Regarding the maximum jitter, it can be stated that running the *RIL* scheduler on a general purpose non-real-time host software system in *GNU Radio* is not sufficient to meet the accuracy constraints of cooperative wireless transmissions.

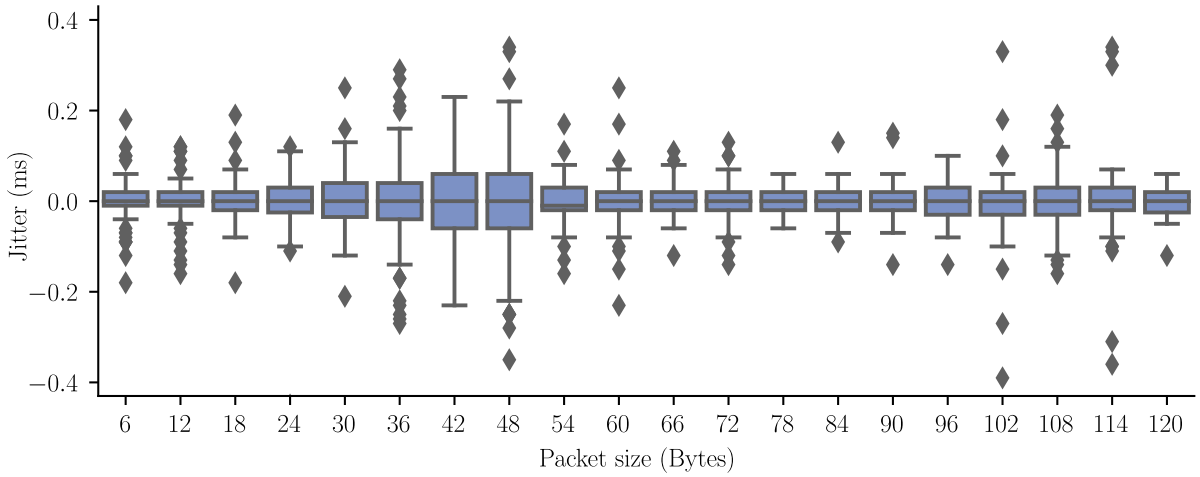
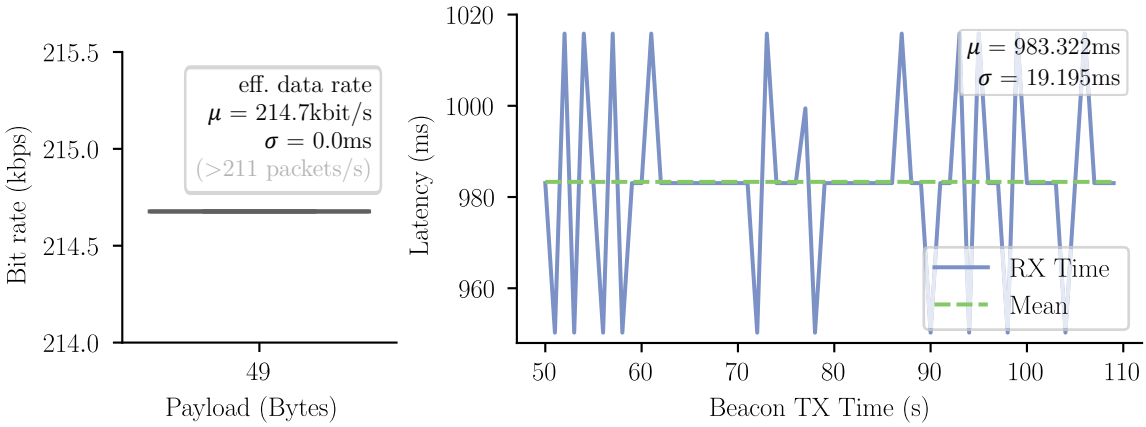


Figure 6.11: Mean value and standard deviation of the scheduling jitter are reported as a function of the *Beacon* frame size.

RF Packet Transmission Throughput and Accuracy

The maximum throughput is determined by switching of the verification of the *PCAP* timestamps in the scheduler routine. Thus, arrived packets with already expired or invalid timestamps in the input stream are immediately forwarded to the corresponding handling routines what ensures continuous packet forwarding. The driver implementation of the *HackRF* requires to continuously write to the entire device buffer before triggering a data transfer. Therefore, the maximum throughput can only be achieved if the device buffer is completely filled with data. Figure 6.12a shows that the hardware is able to guarantee a constant, sufficiently high data rate when using a large packet buffer exclusively, which seamlessly contains several data packets. For example, the effective data rate for the radio packets reaches constant 214.7 kbit s^{-1} what which corresponds to very high packet rate of minimum 211 packets/s for *IEEE 802.15.4* O-*QPSK* 2.4 GHz *PHY* (dependent of the frame size). Furthermore, if the device buffer contains consecutive packets high-precision transmission intervals are possible without measurable deviations.



(a) *TX* data rate maximum (constant bit rate).

(b) Radio packet *RX* accuracy over a *Beacon* transmission period of 60s.

Figure 6.12: Software *RIL* packet transmission performance: *Beacons* are scheduled with **(a)** different sizes, $BI = 0$ us, **(b)** size of 51 bytes, $BI = 983.232$ ms.

With activated offset-scheduling and a corresponding realistic *Beacon* transmission interval between the packets, this high accuracy cannot be achieved out of the box. For the scheduled packets to be transmitted directly at all, the device buffer must be completely filled. We added a *padding* of the output stream to the *SDR* devices buffer size to let a packet immediately been sent to the air but with an additional delay tied to the serial data interface latency and the internal scheduling of the host operating system, which is non-real-time in our virtualized emulation backend. Thus, we can figure out the same effect of fluctuations around the exact mean compared to our evaluation of the accuracy of the chip *RIL*. Figure 6.12 shows a very constant deviation from the mean which indicates a correlation with an unfavorably selected buffer size at the driver.

6.4.3 | Discussion

With this evaluation, the principal feasibility of our two proposed solutions for a *RIL* strategy has been outlined, the respective performance characteristics are identified. Based on these results, we discuss our approach in the following with respect to the requirements identified at the beginning of this chapter. Such considerations of the key requirements have so far been neglected in comparable *HIL* approaches, such as in [124] or [146], for transmissions between a general purpose host system simulator and dedicated external *RF* hardware.

Reactivity: Immediate event detection is implicitly given with the dedicated use of the *RIL* hardware resources whose only task is to detect incoming events. Thus, there are no other parallel tasks running that could limit reactivity regardless of the solution strategy. For example, the reactivity using the chip *RIL* solution is maximal regarding the capacities of the sensor node hardware. For the *SDR* approach, this is also true with respect to the *GNU Radio RIL* process on the host system. Appropriate measurements (ref. Figure 6.9) have shown, for example, that all packets of a burst with a maximum packet rate from standard-compliant hardware were received.

Timeliness: For practical use, the *RIL* scheduling accuracy is not sufficient when running on a virtualized non-real-time emulation backend. However, this is not due to our approach or implementation but to the internal scheduling of the used host operating system. This has been shown by our measurements independent of the *RIL* solution, i.e., accurate timing can be achieved if the models run at least with a dedicated real-time kernel on the emulation host system. Furthermore, this does not exclude a virtualized emulation backend per se, but the achievable scheduling latencies for a concrete real-time test backend must be evaluated in this context (as in [164], for example). Since most computing and server resources are virtualized nowadays, real-time capability in virtualization is a large highly active research field the findings of which are essential to consider for the practical deployment of the *RIL* strategies. Thus, a dedicated emulation host backend at least featuring a real-time core is indispensable for practical use. The deviation under full utilization of the serial interface shows sufficient accuracy of the *RIL* hardware in each case ($\sigma = 0.004\,\text{ms}$ (chip); $\sigma = 0.0\,\text{ms}$ (*SDR*)).

Concurrency: Since via the offset time compensation approach each *RIL* node runs in an independent process, ensuring concurrency is solely driven by an accurate host system clock. The proposed sub-microsecond synchronization of all components in the dedicated *LAN* using *PTP* is state of the art and we did not evaluate it in the context of our approach. On the *ESF* dispatcher backend itself, our approach does not require time-critical execution of parallel *RIL* operations, which provides the greatest flexibility and scalability.

In addition to the discussion of the key requirements, we conclude by highlighting a few more characteristics in relation to the practical use of the two solution strategies.

Concluding Considerations and Solutions Comparison

If the *PHY* implementation is based on a standard-compliant but fixed radio chip hardware, layer functions on the proposed domains are not or only partly adjustable or switchable. Currently available transceiver hardware designs cover only a subset of the functional specifications of the protocol standards, since often various fundamentally different operation modes, frequency ranges, and modulation methods have been standardized. A cost- and resource-optimized hardware implementation cannot achieve this flexibility important in terms of experimentation and evaluation, e.g., for modern cognitive radio approaches.

An important advantage of using sensor node operating systems is that the firmware can be translated for different hardware platforms with only tiny adjustments regarding the serial interface configuration and promiscuous mode activation on the device. Thus, we can provide a great spectrum of common node hardware for the *RIL* transceiver which provides the best representation of the desired characteristics of a specific hardware platform (e.g., energy consumption, *CPU* load, *RX* sensitivity, and many more) compared to the *SDR* approach. Our chip implementation for the ATmega128RFA1, for example, is particularly suitable for the *RoSeNet* emulation hardware. The transceiver firmware can even run as a process on standard Linux systems, enabling virtual testbeds and integration into a pure virtual emulation (cp. simulators and virtual testbeds alongside sensor node operating systems, introduced in Subsection 2.3.1). This feature allows for doing tests with multiple attached virtual nodes via so-called *Pseudo Terminals* (cp. stream-connection interfaces in Subsection 5.3.3).

The *GNU Radio SDR RIL* scheduling is applicable to other link layer protocol standards and transceiver hardware. As common implementations for *SDRs* focus only on the modulation of the protocol standard, our approach shows that we can gain full control of all *PHY* features and parameters, modeling the *SDUs* instead of transmitting only modulated *MAC* frames. Moreover, further general parameters and different modulation schemes, as well as *PHY* specifications according to the latest revisions of our reference protocol specification *IEEE 802.15.4* can be added straight-forward. Compared to dedicated hardware parameters for chip transceivers, the *SDR* measurements also show that in order to determine the optimum hardware parameters on a general purpose hardware platform for the intended application environment, additional measurements in special environmental conditions are necessary. This is also required, for example, to provide a better coverage of the value range of the energy detection. Furthermore, based on the signal strength measurements of the packet reception (ref. Figure 6.9), it is obvious how, for example, the influence on the channel quality can be emulated via various device parameters. Thus, the software *RIL* solution will eventually enable flexible experimentation and *PHY* modeling diversity in network scenarios with regard to the simulated upper layers of the protocol stack to support the evaluation in cognitive radio sensor network research and development.

Running specific higher layer protocol stack implementations directly in *SDR* environments is also possible, but it is often limited and does not provide sufficient experimentation flexibility (e.g., no variations according to parameter settings, control algorithms, etc. are possible). For example in [161], the IEEE 802.15.4 *MAC* is severely limited (e.g., no medium access control algorithms) and the network stack is taken from a hardware-specific lightweight implementation. Nevertheless, host-based *Software RIL* with *GNU Radio* can also be applied for injecting controllable noise and technology interference at the into the *RF* network of the analog channel emulation. This offers great capabilities of studying wireless networks in dense or interference-impacted radio environments.

An advantage of the *SDR* solution is that only *IEEE 802.15.4* payload data needs to be transmitted to the hardware, whereas complete *PCAP* packets are evaluated on the hardware with the *RIL* chip. Investigating in future enhancements of the software *RIL* transceiver in *GNU Radio*, we propose a permanent data transmission on the serial interface to increase the accuracy based on continuous (interrupt-free) streaming by filling transmission-free periods with zero data. This requires an exact calculation depending on the *RF* modulation and frequency, as well as the data rate and buffer size at the serial interface and customizing the device driver implementation. However, this approach could largely decouple the transmission accuracy from the operating system scheduler.

Figure 6.13[10] summarizes the two solution approaches based on several discussed properties in overview. The main benefit of our *SDR RIL* implementation is the *PHY* modeling flexibility, whereas, on the other hand, the ease of use and experimentation (standard conformity) of the chip hardware is suitable for many application areas. In conclusion, the two approaches are beneficial for the *Split-Protocol-Stack* and should be considered when setting up parallel protocol simulation and radio emulation.

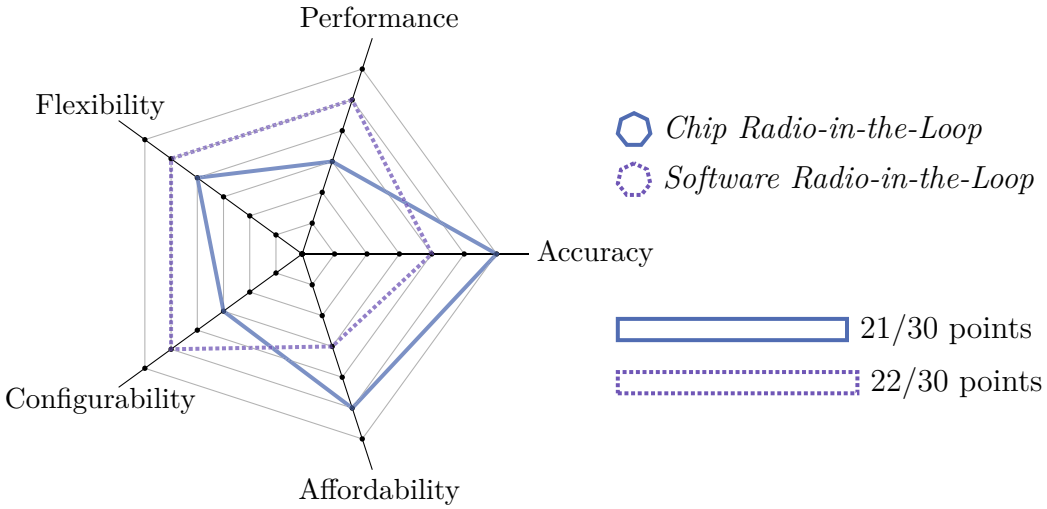


Figure 6.13: Comparison of the Chip and Software Radio-in-the-Loop Solution[10]

[10] The exact interpretation of the radar plot evaluation is given in Table A.1 in in Appendix A.

7 | Radio-in-the-Loop Channel Emulation

The main advantage of *RIL* wireless transmissions is the interfacing with the physical radio environment to enable emulation on the radio channel through virtually generated network traffic. Emulating the wireless communication according to a testbed-like, analog radio evaluation system is certainly an electrical engineering domain, where a lot of know-how in *RF* hardware and physics is required. Nevertheless, there are some important details and considerations in terms of the *Split-Protocol-Stack* wireless emulation methodology that belong to the modeling and algorithmic domains.

In the following, we present strategic details of the radio channel modeling for *RIL* wireless transmissions. We particularly address the need to consider channel emulation modeling in high-level network architecture and topology planning. In this context, we contribute with reference scenario measurements and significant insights of an optimization approach to solve the algorithmic problem of mapping an abstract wireless network evaluation scenario to an allocation of hardware resources for a particular radio topology in a *NET*. The chapter concludes with a discussion of the fundamental achievements in the context of capabilities for enhancing *RIL* hardware channel emulation.

7.1 | Radio Channel Emulation Modeling Strategy

The main focus of the *analog radio domain* for the *Split-Protocol-Stack* strategy (level ③ in Figure 4.2) considers the environmental scenario modeling. Our first suggestions about generic solutions for the generation of evaluation scenarios for *NETs* were discussed based on our research statement in 2015 [165] in the context of the radio channel emulation platform *RoSeNet*. One of the most important unsolved research questions and also the main motivation behind our investigations have been obvious: *How to utilize a cabled and controllable RF environment for a simulation-driven network evaluation scenario?* The technical challenge is to transform analog radio emulation hardware into a discrete-event system which features an abstract and time-accurate access control scheme to the emulation hardware resources. However, the configuration challenge is that the channel emulation modeling must match the scenario of the network and protocol simulation system. Thus, each node in the simulation requires an accurate hardware representative in the *analog radio domain*. With our emulation modeling strategy, we assume that *RIL* transceiver hardware itself is involved in emulating physical and radio channel effects in the *RF* environment.

7.1.1 | Wireless Network Planning

For the topological structure of a wireless network, it is essential to distinguish between two different terminologies. The *radio topology* refers to the physical placement and reachability between wireless nodes on the *PHY*, whereas the *network topology* refers to the logical link connectivity. An intended network topology imposes direct requirements for the radio topology which must ensure adequate wireless connectivity with appropriate radio link characteristics. Nodes that are physically reachable via the radio interface do not necessarily have higher-layer network connectivity, but must still be considered when planning the radio topology, since their participation in the radio channel according to their media access has a direct impact on the overall network performance. Of course, any kind of higher-layer network topology is possible if all nodes are physically connected with each other. This is usually beyond the scope of resource-constrained wireless multi-hop networks. In contrast to wired networks, where the network topology is mainly based on functional, security, and management aspects, the planning of wireless networks is more focused on the radio characteristics in the operating environment.

Wireless network planning is a generic term and a complete own strategy to determine the radio topology and configuration parameters for a specific application in a deployment environment. Regarding this strategy, high-level considerations and performance parameters with respect to the physical node placement are summarized in a basic conceptual analysis in the following. The *Split-Protocol-Stack* helps to exactly consider these properties in physical accuracy when planning a wireless network. The most important high-level network considerations regarding wireless network planning are summarized using the following keywords:

- **Security** considerations beyond node-individual and protocol-specific cryptographic methods are impacted by the radio topology and thus increase the complexity of node placement to a significant degree. For example, accurate node placement and environmental monitoring can aim at *authenticity* by creating a distinct environment in which an attacker generates an unambiguously different *radio fingerprint* than trusted nodes. This has positive implications on attack detection and prevention (cp. [166] for a summary of implications).
- **Extensibility** of the wireless network is not always a mandatory requirement, but it becomes an important design aspect, especially in mobile ad hoc networks. This means, that the addition or spontaneous appearance of new nodes must not lead to congestion, e.g., of a spatially limited radio environment or cell.
- **Administration & Maintenance** are operational features that also must be considered well in terms of the physical access to the device hardware, but in particular to ease heterogeneous network configuration. *Topology maintenance* and *topology control* are often mentioned regarding different triggering aspects (e.g., energy efficiency, lifetime, and reliability).

In contrast, certain performance parameters regarding *PHY* radio transmissions must be determined. Since these parameters often contradict each other during planning (e.g., tradeoff between throughput maximization and energy efficiency), ideal characteristics are to be achieved for the overall network operation. Optimization approaches are increasingly applied for this purpose. The most important performance parameters for radio topology planning, for each of which an optimization approach is given as example, are:

- **Link Capacity & Throughput** are often mentioned in the context of node placement to maximizing the performance in a bandwidth-limited network [167]. For example, *capacity graphs* are used to figure out suitable radio topologies, e.g., gateway node placements to avoid too much nodes on the same channel.
- **Energy Consumption** in resource-constrained wireless networks is the overall most considered optimization target. Maximizing network lifetime by minimizing the average energy consumption in the entire network (e.g., based on *Mixed Integer Non-Linear Programming (MINLP)* in [168], or energy-efficient routing) is beyond the initial placement but must be considered at this point. Key issues, such as the equal distribution of nodes, fairness in timesharing the medium access, co-channel interference, or the utilization of antenna characteristics play a crucial role.
- **Reliability & Resilience** are to be particularly considered in control processes with high interference potential and moving objects (e.g., robot arms). Fault-tolerant radio coverage and connectivity to ensure reliability is achieved not only by providing backup link capacity using multi-technology and multi-channel but also by recovery and reconfiguration based on base-station optimization (cp. [169]).
- **Timeliness & Latency** are becoming increasingly important in time-sensitive *WSNs* and wireless real-time networks for control and monitoring applications. The optimal sink node placement [170] and gateway relocation [171] are, for example, two objectives for a latency-related optimization regarding wireless network planning, e.g., to ensure that data reach the sink with a maximum latency of certain milliseconds.

To apply an optimization method the first step is to find a suitable network and environment model. This requires a deep understanding of the effects of wireless communications in general and extensive knowledge of the link layer radio technology used in particular. For difficult operating environments, e.g., with strong interferences, on-site measurements are inevitable and must be taken into account in the planning. Practices, such as *heat mapping* (e.g., using signal strength measurements) and considerations about materials, walls, windows, shelving, elevators, and stairwells for indoor environments as well as humidity, vegetation, or slope for outdoor environments must be included in the evaluation. We carried out such reference measurements (ref. Section 7.2) in a particular outdoor environment as a basis for further experiments. The wireless network planning question was: *What types of nodes and associated technological characteristics are required for full radio coverage and what basic network topology is optimal.*

7.1.2 | Radio Topology and Channel Emulation Modeling

The result of wireless network planning are specific network architectures, technologies, device types, topologies, as well as environmental and radio performance parameters. These properties must be carefully and accurately mapped to the emulation system. Therefore, a main focus of this chapter is the conceptual analysis for the configuration and implementation of a target radio topology on a radio channel emulation system or the *NET*. Furthermore, the representation of channel effects within the emulation system is discussed according to Subsection 2.2.1.

Radio Topology and Large-Scale Fading Emulation

To determine the *large-scale fading* between individual wireless devices (ref. Equation 2.1) two-dimensional coordinates of the modeled scenario and the *RX*/*TX* gain parameters of the *RF* hardware form a basis for the network scenario model. The signal fading between nodes can be derived from reference measurements or calculated using, for example, a simplified uniform spread of energy in free space given by a path loss model. Common models provide idealized results that often do not correspond to the exact physical behavior (ref. Section 7.2), but which are suitable to approximate a planned topology in a real test setup. An obvious way of modeling the radio topology is the abstraction using graphs, comparable to modeling the network topology (ref. Subsection 2.2.5).

A *Unit Disk Graph (UDG)* (also known as *intersection graph* of coplanar congruent disks) is a graph $G = (V, E)$ whose *vertices* V can be mapped to numerical coordinates in a cartesian coordinate system in the plane and whose *edges* E are defined by pairs of coordinates with uniform Euclidean distance from each other. Figure 7.1 shows an example *UDG* representation for modeling the radio topology based on physical positions. Conversely, not any network graph G can be mapped as *UDG*. The problem of recognizing whether a given graph is a *UDG* is $\mathcal{NP}$-hard [172].

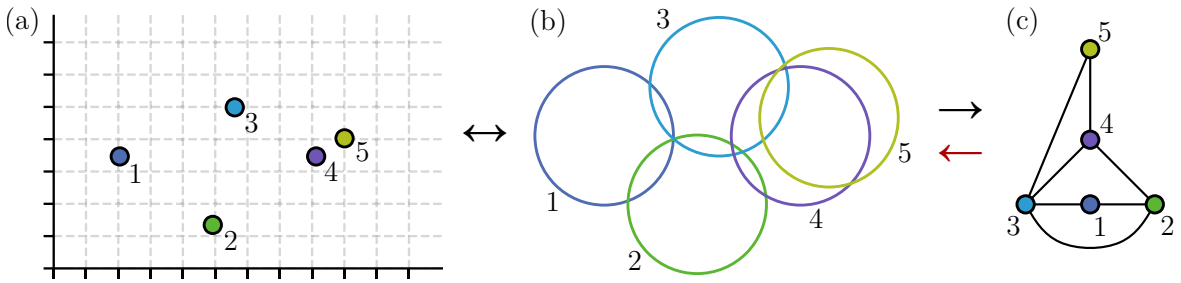


Figure 7.1: Example *UDG*: **(a)** realization of G; **(b)** coplanar congruent disks for G; **(a)** model $G = (V, E)$. All representations correspond exactly to the same physical radio topology of a wireless network.

Taking the context of *UDGs* for modeling into account plays a significant role, since not any arbitrary topology (node placement) is possible in terms of radio connectivity between pairs of nodes. Besides physical node placement, a channel emulation platform allows to *arrange* testbed nodes to virtual geographic positions. When considering the radio topology in a wired *analog channel emulation* system, variable attenuation within the signal path can be configured to form a radio topology (cp. selected analog link emulation approaches in Section 3.3). Comparable to the above mentioned problem, the challenge of allocating hardware resources on an analog radio *NET* for emulating the graph-based radio topology is also $\mathcal{NP}$-hard, but a solution differs significantly from finding physical coordinates. We present our analysis and solution approach for this channel emulation requirement in Section 7.3.

Mobility Emulation

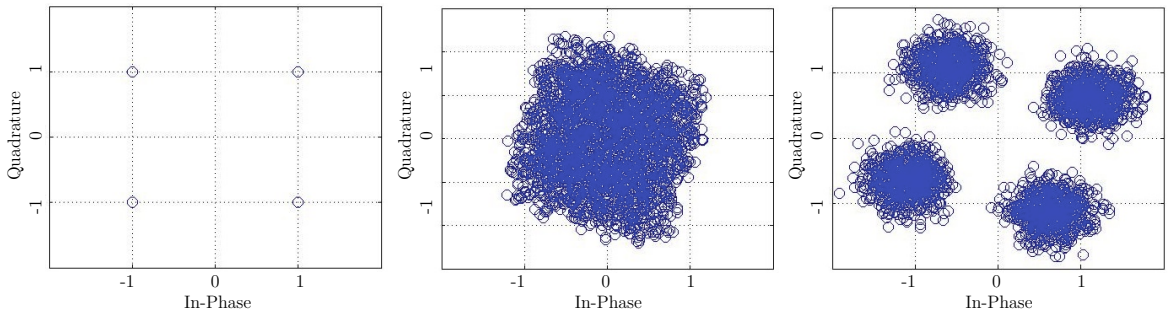
Mobility emulation is primarily based on mobility models of an evaluation scenario, for which different approaches can used to calculate node positions (ref. Subsection 2.2.5). Thus, mobility model calculations trigger a variance of *RSSI* values which directly affect the physical radio topology and require continuous re-calculations of the configuration parameters for virtual node positions on the emulation system. For example, the signals enter the propagation interference range above a certain threshold of signal strength attenuation depending on the protocol standard (ref. *PHY* transmitter and receiver parameters in Table 2.1) and the device hardware. With reference to the *UDGs*, mobility eventually causes a transformation of the intersections of the congruent disks that can eventually eliminate or create vertices in G. In the configuration's worst case, this means that several nodes in the emulation system would suddenly have to be allocated differently during the emulation runtime. Furthermore, it is not necessarily feasible to change a node position in the configurable *RF* signal path of the emulation system at all (cp. Section 7.3). Our strategy is therefore to enable pseudo-random mobility and mobility based on the node's gain parameters and antenna properties, as introduced next.

Antenna Emulation

The characteristics of the antenna affect the attenuation of all other nodes depending on the physical position in the scenario and increase the mapping complexity of a node placement on a testbed hardware allocation (intensity of the signal fading as a function of the direction of transmission). In a first analysis step we focus on the simplistic model of an isotropic antenna that additionally impacts the transmission and reception gain of a node, as introduced with Equation 2.4 in Subsection 2.2.1. Since a wireless transceiver is able to adjust its transmission power and the receiver sensitivity, the antenna modeling is accounted to the *PHY* model parameters `phyTransmitPower`, `phyTXgain`, and `phyRXgain` (ref. Section 6.2 and Section 6.3).

Emulating Channel Effects, Interference and Noise

Due to *reflection*, *scattering*, or *diffraction*, the signal propagates along multiple paths with different attenuations and delays from the source to the destination mainly causing the temporal dispersion of the signal. Emulating these effects based on analog channel emulation depends on special *RF* hardware components in the signal path (e.g., delay lines, frequency synthesizer, frequency-selective attenuators, filters, and many more), or *ray tracing* techniques in simulative digital channel emulation (cp. [173]). On the other hand, host/node based *equalization* refers to techniques which mitigate the transmission effects on the radio channel at the radio device hardware, e.g., *multipath channels* lead to interference effects at the receiver (e.g. *Inter Symbol Interference (ISI)*) that make signal reception difficult. Since a well-designed *equalizer* can compensate multipath effects and restore the expected *signal constellation* (e.g., based on simulation in *MATLAB/Simulink* [174] or *SDR* in *GNU Radio* [175]), *SDR* can also be utilized to *create* interference in low-noise environments of analog channel emulation. Figure 7.2 shows example constellation diagrams of *QPSK* symbols before **(b)** and after **(c)** equalization. Our strategy is to virtually emulate channel effects based on amplifying or distorting signal components on the host with *TX* equalization at the *SDR RIL* transceivers in *GNU Radio* (ref. Section 6.3).



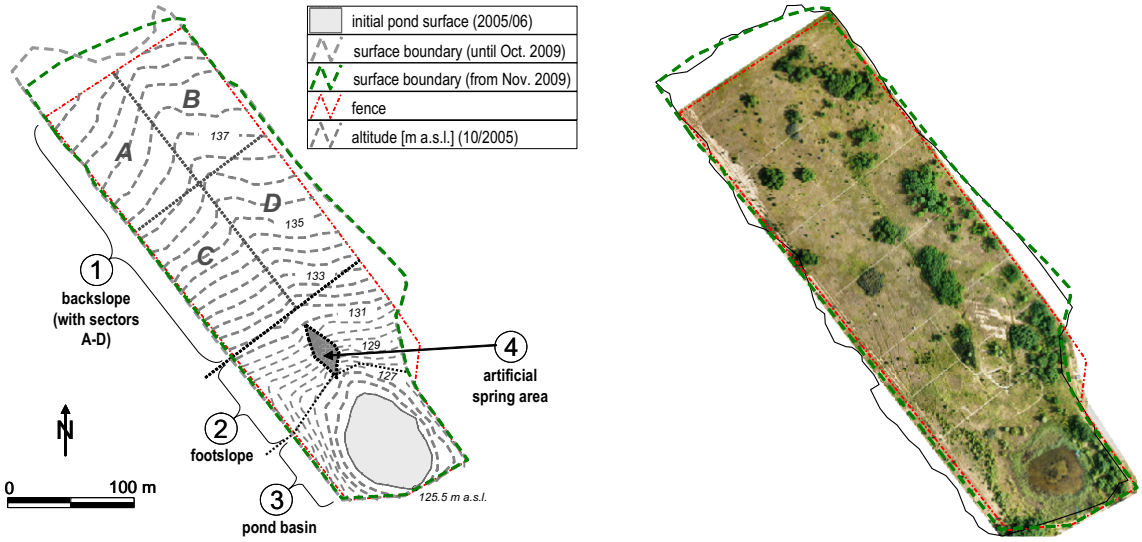
(a) Transmitted symbols (b) Received symbols (c) Equalized symbols

Figure 7.2: Example constellation diagram of an adaptive channel equalization based on *QPSK* symbols in *MATLAB/Simulink*, simulation results taken from [174]

Furthermore, also technological *interferences* and background *noise* injection are application areas of host/node based emulation with *SDR* transceivers in the *Split-Protocol-Stack* approach. *GNU Radio* application frameworks, for example, offer a wide spectrum of either implementations of link layer wireless technologies and signal processing components to add noise sources. According to Equation 2.5, background noise power σ^2 affect the measurable *SNR* for the emulated wireless transmissions, which is fully configurable with virtual components regarding the amplitude and type of noise. Based on this strategic considerations, the *RIL* channel emulation is able to model the key features and effects of real wireless transmissions.

7.2 | Chicken Creek Reference Measurements

The artificial catchment *Chicken Creek* [176] is a small area of a former open-cast lignite mine in Lusatia, Germany, the name of which derives from a small river (the *Hühnerwasser*) which was destroyed during this mining in the 1980s. The research objective at this site has been to study the initial phase of an ecosystem development using a model system. Thus, basic monitoring equipment can indicate the health and properties of the soil (e.g., groundwater gauges, atmospheric deposition samplers, soil moisture probes, weather stations). In this context, remote sensing can greatly facilitate the process of data acquisition and evaluation of ecosystem monitoring. We performed some basic reference measurements in the backslope (① in Figure 7.3a) to evaluate the deployment of *IEEE 802.15.4* as wireless monitoring technology and equipment.



(a) Chicken Creek division in sectors [176] (b) Chicken Creek vegetation condition[1]

Figure 7.3: Chicken Creek ecosystem monitoring general map

As in Section 7.1, the need for heat-mapping and on-site reference measurements in the wireless network planing process has already been pointed out. These measurements serve for a realistic reference scenario and hands-on use case for various issues in the context of the *Split-Protocol-Stack*. Central issue was to enable a simple, cost-effective, automated, wireless acquisition of all digital measurement points in the terrain to a central location, as well as the provision and visualization of all values on a common platform[1]. High-level network considerations are rather less relevant for this monitoring scenario (not security-critical operation in the field for years without planned extensions).

[1] Chicken Creek catchment data portal (aerial photo): `https://www.b-tu.de/chicken-creek/apps/datenportal/`

7.2.1 | Measurement Setup and Node Placement

Reference measurements of signal strengths between pairs of nodes were carried out in sectors A-D (backslope) (ref. Figure 7.4). The soil pit *SC* was selected as the central receiving node because of the nearby solar-powered weather station *W*1 which provides a continuous power supply. In our experiments, we used standard sensor node hardware from *Microchip* (ATmega128RFA1[2]). We set up transmitter and receiver firmware implemented based on *Contiki* to either transmit bursts of radio packets (50 each) or receive *IEEE 802.15.4*-compliant telegrams and record *RSSI* values for each packet received. The positions of the nodes were selected firmly on the basis of important measuring points as in Figure 7.4 and placed approx. 1.5 m above the ground level in each case. The reception range of *SC* is highlighted as 100 m, since this specification is often given as a guideline for the maximum range of *IEEE 802.15.4* device hardware in the outdoor environment. Of course, the actual range depends not only on the environmental conditions but also on the actual hardware used, the transmitting power, the sensitivity of the receiver, and the antenna characteristics.

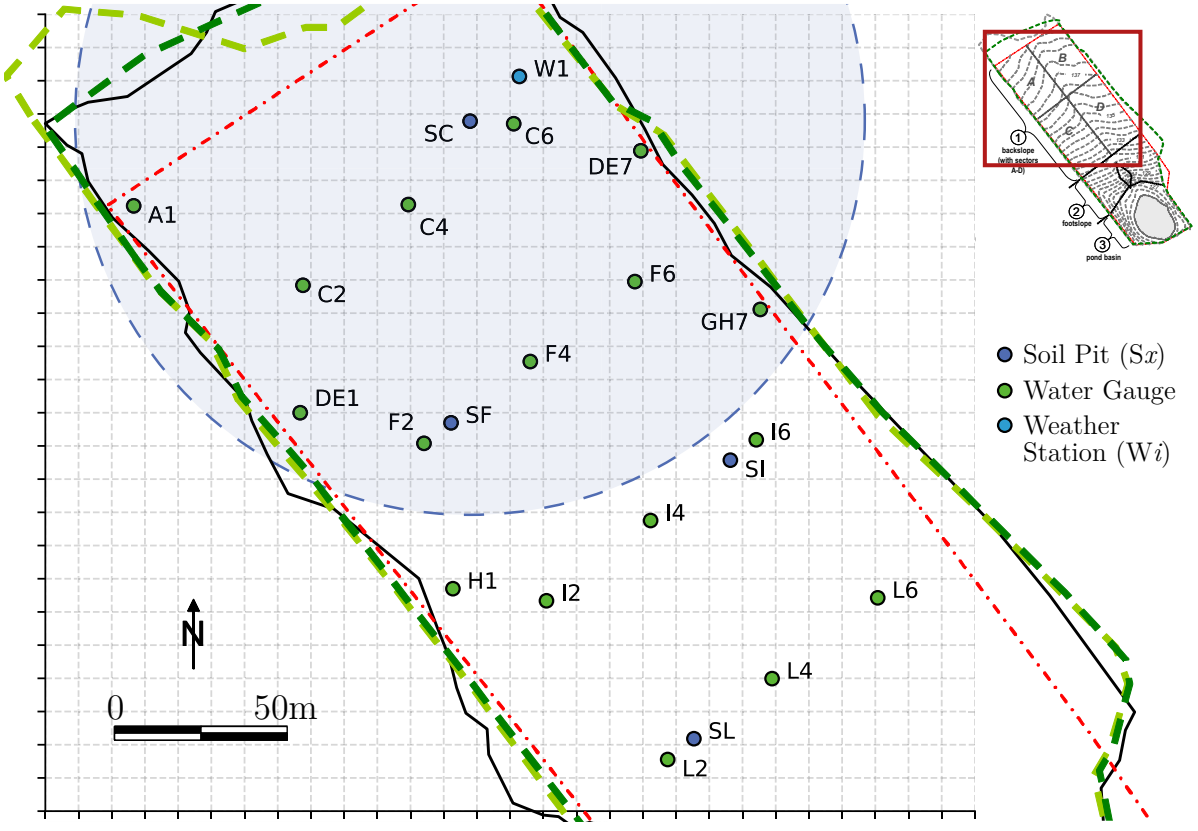


Figure 7.4: Reference coordinates for the *RSSI* measurements in the *backslope* (sectors A-D). *W*1 was set up as receiver for all wireless transmissions.

[2] ATmega128RFA1: `https://www.microchip.com/ATmega128RFA1`

7.2.2 | Measurement Results and RIL Emulation Relevance

Figure 7.5 shows the measurement results based on the received signal strength at the fixed receiver (*SC*) as a function of the transmitter distance compared to the calculated theoretical values. The performance parameters correspond to the maximum values of the hardware used (*RX* sensitivity: -100 dBm, *TX* power: +3.5 dBm). The theoretical values are based on a simplified uniform spread of energy in free space (ref. Equation 2.1).

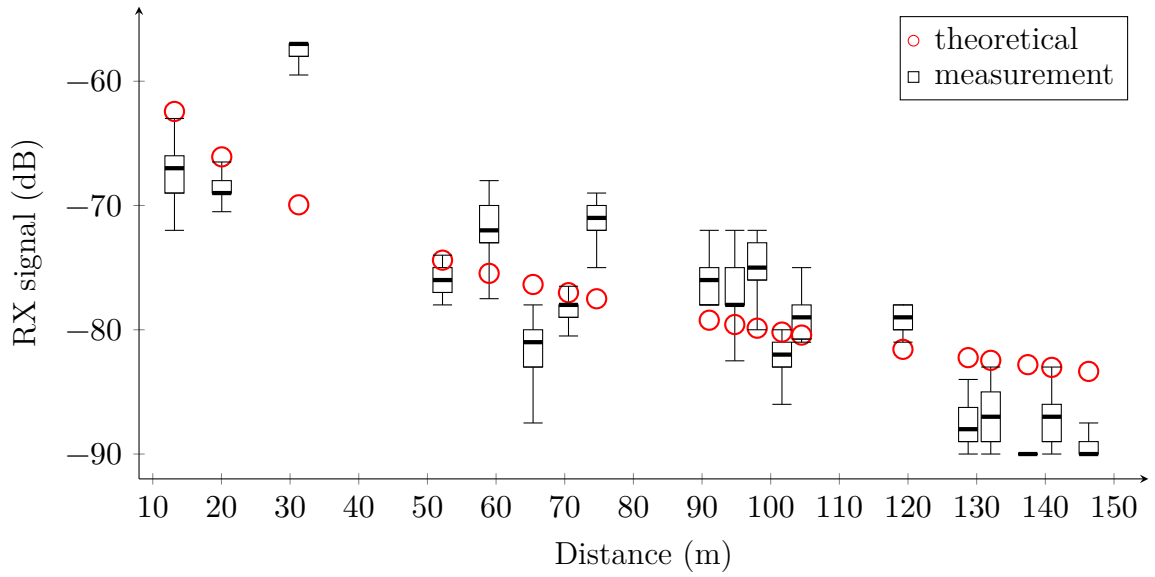


Figure 7.5: RSSI measurement results

The measurements show an overall reasonable radio coverage but at the same time strong deviations from the theoretical reference value and violent differences of similarly distant measurement points (e.g., 58-68 m in Figure 7.5) up to ≈ 20 dB. Furthermore, strong fluctuations of the received signal strength of one and the same measuring point in the range of up to 10 dB could be measured. These deviations and fluctuations are expected to be even more pronounced throughout the year due to leaf cover, humidity, and solar exposure. From these results follow that planning solely based on model calculations and coordinates is not a reliable basis, even in outdoor areas due to slope and vegetation. Nevertheless, there is a significant potential for energy savings with adaptive signal transmission power and reception sensitivity as well as the use of antennas with directional characteristics (cp. antenna impact in Subsection 7.1.2). Besides the conceptual derivation of a network architecture for the monitoring system (cp. [177]), these measurement results provide reference data for evaluating adaptive and cognitive radio transmissions based on the *RIL* channel emulation. This evaluation scenario of optimizing node energy consumption for *RX*/*TX* in terms of reliability and radio coverage is presented as an example using the *Split-Protocol-Stack* emulation in Section 8.2.

7.3 | Hardware Allocation in Network Emulation Testbeds

In low-scale test setups with a problem-specific focus, the system architecture results from the *RF* planning of the particular target scenario (e.g., [111] from the analysis in Section 3.3). Flexible large-scale emulation on the *NETs*, on the other hand, depends on appropriately configuring the signal path according to the emulation system architecture. The approach towards hardware allocation was first introduced in [55] and is built upon a graph-based model definition including an abstraction of the exemplary *RoSeNet* (ref. Section 3.3) hardware resources.

7.3.1 | Problem Statement and Model Definition

Network simulators allow one to create arbitrary network topologies. Within the methodology of the *Split-Protocol-Stack* approach, appropriate hardware resources must be allocated for a given scenario, and *RF* configuration parameters must be determined. The basic idea and problem is known as *Network Testbed Mapping Problem*, which has often been discussed in the context of the dynamic network configuration for *Network Virtualization* on wired networks and testbeds ([178–180]). So far, virtual links and overlay networks require optimization of inter-node bandwidth, latency, or routing requirements in *LAN*s. In the field of analog channel emulation testbeds, the problem differs significantly, as a consideration of the signal strength on the link between nodes must also be taken into account and traffic on links cannot be effectively routed as by using packet switching technologies.

By using *RoSeNet*'s channel emulation architecture for our prototype, we are able to emulate a large-scale fading of signal transmissions in the *RF* network which eventually allows the isolation of individual nodes (ref. *RoSeNet* hardware architecture in Figure 3.5). One important constraint is that *RoSeNet* has both fixed (architectural, non-adjustable) and variable signal attenuators (adjustable by configuration parameters) in its coaxial environment. Selecting testbed nodes based on a *RF* configuration results in a network with a certain *virtual* radio topology. The reverse way of deriving a selection of nodes and a setting of attenuators from a virtual topology (e.g., a *UDG* of a network scenario) turns out to particularly be complex, similar to the distance geometry problem (cp. [172], [181]). At the same time the quintessence and problem is to create radio topologies that are possible to set up in the simulation and the radio emulation domain because not any arbitrary attenuation between pairs of nodes can be configured according to the emulation system constraints. To solve this problem the first step is to specify a suitable data structure for the scenario and the *NET* architecture. This is practically presented and tested in the exemplary realization based on *RoSeNet* next.

Abstraction of the Radio Topology

The modeling of *RF* topologies by graphs was introduced with *UDGs*. Considering the transmission topology of interconnected nodes, with respect to the exact attenuation values between pairs of nodes, a *UDG* expands to include weights on the edges (a_e). We refer to such a graph as an *undirected attenuation graph* G. The radio topology of a test scenario is called *scenario graph* $G_S = G$ (ref. Table 7.1). The attenuation is usually specified in dB with integer values. The scenario graph in Figure 7.6 serves as practical example for demonstrating the allocation process. Regarding the *Chicken Creek* reference measurements in Subsection 7.2.1, this scenario corresponds approximately to the arrangement of the nodes $SC - C6 - W1$ in Figure 7.4.

G:	undirected attenuation graph,	$G = (V, E)$
	set of nodes,	$V = \{v \mid v \in N\}$
	set of weighted edges,	$E \subseteq \{(i, j, a) \mid i, j \in V; a \in \mathbb{N}\}$
G_S:	scenario graph	$G_S = G = (V, E)$
a_e:	attenuation value of the edge	$e \in E(G_S)$, $a_e \in \mathbb{N}$

Table 7.1: Variable definition for the scenario radio topology

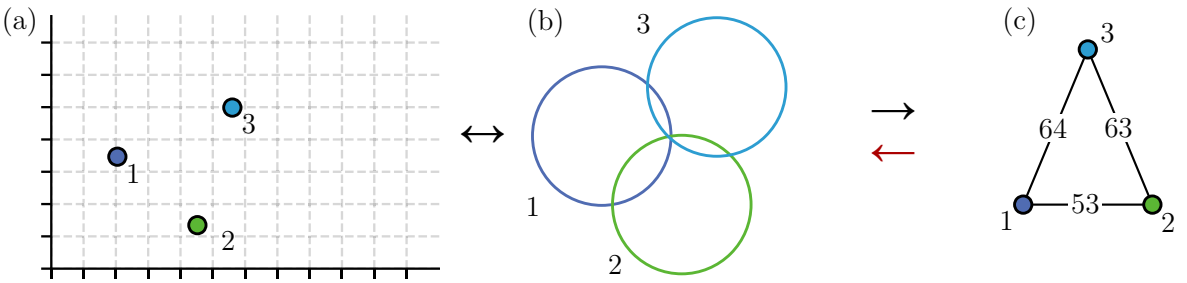


Figure 7.6: Example scenario graph: **(a)** node coordinates of G_S; **(b)** coplanar congruent disks for G_S; **(a)** attenuation graph $G_S = (V, E)$.

Abstraction of the Radio Emulation Hardware

A *RIL* testbed architecture with coaxial-based radio links can also be represented as an undirected attenuation graph G. With *RoSeNet*, we have a plain tree structure (ref. Figure 3.5a) in which the root is the central *anchor node* connected to *chains*. Each chain has several modular entities, called *panels*, which include the actual sensor nodes (the tree leaves). A single *emulation panel* in turn can also be described in its internal structure using this graph abstraction. A special feature is the *attenuator* module for which the graph contains different paths depending on the exact attenuation values.

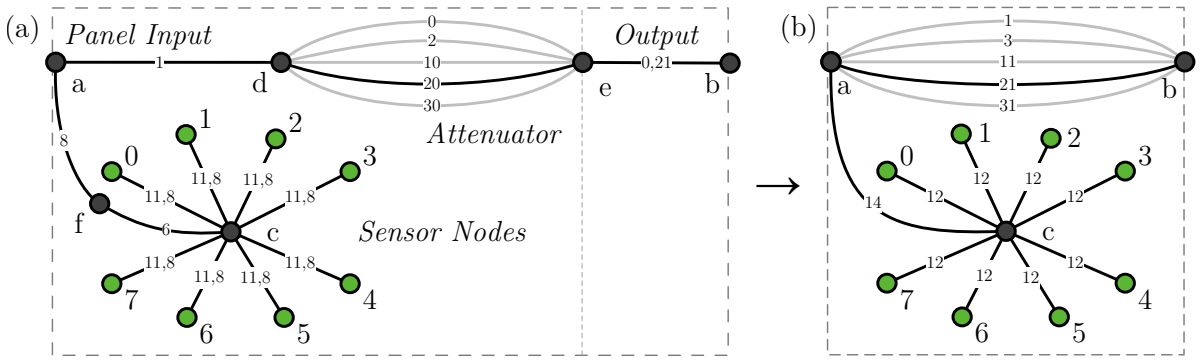


Figure 7.7: *RF* network representation of a single emulation panel with up to eight sensor nodes *(b corresponds to the input of a neighboring panel, c to the* RF *signal splitter to all nodes)*. With the *attenuator* module, we can *select* a single signal attenuation edge between adjacent panels. To further facilitate this abstraction, architectural-fixed attenuation (*d*, *e*, *f*) is included in the controllable attenuation and rounded to integer values in **(b)**.

As depicted in Figure 7.7, we modeled the graph-based abstraction, the *RF* dependencies, and the node types of the target emulation hardware for our allocation scheme. The whole emulation *RF* network consists of the connection of several *panels* in *chains* according to the actual arrangement which is referred to G_H. The *anchor* node corresponds to the root power splitter (start of each panel chain) in Figure 3.5a. An overview of algorithmically important variables and parameters of the model definition is given with Table 7.2.

N:	a set of hardware node types,	$\{'ATmega128RFA1', ...\}$
H:	a set of panel *RF* node types,	$\{'anchor', 'splitter', 'input'\}$
A:	a set of static attenuation values,	$\{1, 3, 11, 21, 31\} \subseteq \mathbb{N}$
G_H:	undirected hardware graph,	$G_H = G = (V, E)$
	set of nodes,	$V = \{v \mid v \in \{N \cup H\}\}$
G_A:	undirected allocation graph,	$G_A \subseteq G_H = G = (V, E)$
	set of nodes,	$V = \{v \mid v_r =' anchor', v_l \in N\}$
G_P:	undirected allocation path,	$G_P \subseteq G_A$
	set of nodes,	$V = \{v_0, ... v_k \mid v_0, v_k \in N, v_i \in H\}$
P:	a set of connected hardware panels,	$\{p \in V(G_H) \mid p =' input'\}$
P_e:	a set of hardware panels in path	$e \in E(G_S)$, $P_e \subseteq P$
ϵ:	deviation of attenuation values,	$\{\epsilon \in \mathbb{R}_+ \mid \epsilon < 1\}$
a_i:	attenuation value of panel i,	$i \in P$, $a_i \in A$
x_{ij}:	edge usage of panel i,	$i \in P$, $j \in A$, $\{x_{ij} \in \mathbb{N}_0 \mid x_{ij} \leq 1\}$
a_{ij}:	edge attenuation values of panel i,	$i \in P$, $j \in A$, $a_{ij} \in A$

Table 7.2: Variable definition for the panel hardware

7.3.2 | Allocation Approach

The allocation objective is defined regarding the aforementioned model-based abstraction. What is searched for is an attenuation graph G that results from the configuration of the emulation hardware parameters, so that the scenario graph G_S is a subgraph of the hardware graph G_H of the emulation system in a tolerance range with respect to the edge weights (ref. postcondition (2) in Algorithm 6). The resulting allocation graph G_A represents the associated configuration of the involved hardware and enables automatic allocation. We have developed a multiple stage allocation process to find an appropriate configuration of the *RF* signal path on testbed platforms (exemplary applied to the *RoSeNet* hardware model G_H). It is based on graph analysis (4) and linear optimization (7-12) with the allocation procedure given in Algorithm 6.

Algorithm 6: Node Hardware Allocation Procedure

```
(1)  Precondition : |scenario nodes| ≤ |hardware nodes|
(2)  Postcondition : G_A ⊆ G_H with tolerance to a_e
(3)  generate G_S and G_H
(4)  I_λ = analyze scenario G_S
(5)  while (true) do
(6)      G_A = assign nodes from G_H with G_S and I_λ
(7)      Function calculate_attenuation(G_A, G_S)
(8)          make LP from |V(G_S)| and |V(G_A)|
(9)          add LP scenario constraints from G_S
(10)         for ϵ ∈ Set_ϵ do
(11)             add LP hardware constraints from G_H with ϵ
(12)             solve LP
(13)             if attenuation found then
(14)                 allocate graph G_A
                     return
                 end
             end
         return
     end
```

The allocation process includes multiple steps, starting with the generation (3) 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_λ) of the scenario graph are calculated and processed by an initial node allocator (6)which makes a decision regarding the panel-node placement G_A. In the third step, it transfers the graph representation with the allocated nodes and all additional constraints (coming from the hardware and the *RF* dependencies) to a *Linear Program (LP)* which can then be

solved using *MILP* (7). If a solution is calculable the algorithm is terminated, otherwise the parameters are adjusted (10) (e.g., the deviation from attenuation parameters ϵ) or the algorithm must calculate a new panel-node placement G_A. If no solution can be calculated at all (e.g., due to hardware constraints), the scenario is unsupported and the initial radio topology must be changed.

Mixed Integer Linear Programming Model

For the second step of the problem analysis, it is assumed for simplicity that a distribution of panels and nodes is already known but the configuration of the attenuation parameters is required. To solve this problem we propose an approach based on the *MILP* optimization model. The required attenuation parameters in certain interval limits are represented by a cost function The two attenuation graphs G_S and G_A (e.g., the distribution of panels and selection of nodes) are described by a set of constraints in the form of linear equations and inequations. The attenuation constraints are listed below for the selected hardware panels and the scenario.

MILP-Constraint 1: *Each panel $i \in P$ can have only one single attenuation value.*

$$\sum_{j \in A} x_{ij} = 1 \text{ (the sum of the edge usage of panel } i \text{ is equal to 1)}$$

MILP-Constraints 2-3: *Each panel $i \in P$ must have an attenuation within limits.*

$$\sum_{j \in A} (1 - \epsilon) a_{ij} x_{ij} \leq a_i \text{ (attenuation lower bound)}$$
$$\sum_{j \in A} (1 + \epsilon) a_{ij} x_{ij} \leq a_i \text{ (attenuation upper bound)}$$

The *MILP*-Constraints 2-3 guarantee that the attenuation values a_{ij} does not deviate upwards or downwards by more than the value defined by the deviation ϵ. Note that *MILP*-Constraint 1 causes only a single attenuation value to be summed up.

MILP-Constraint 4: *Each edge $e \in E(G_S)$ must set attenuation values in a path.*

$$\sum_{i \in P_e} a_i = a_e \text{ (panels in path between two nodes)}$$

The *MILP*-Constraint 4 guarantees that the sum of attenuation values in a set of panels in a path between two nodes corresponds to the attenuation value for each edge $e \in E(G_S)$ (ref. (9) in Algorithm 6). A graphical representation of an allocation path is highlighted in Figure 7.9a. This means that the number of path constraints for a particular scenario is given according to the number of edges in the scenario graph S_G.

$$\begin{pmatrix} a_{11} & \cdots & a_{1i} & x_{111} & \cdots & x_{1ij} & b_1 \\ \vdots & & & & & & \vdots \\ a_{n1} & \cdots & a_{ni} & x_{n11} & \cdots & x_{nij} & b_n \end{pmatrix}$$

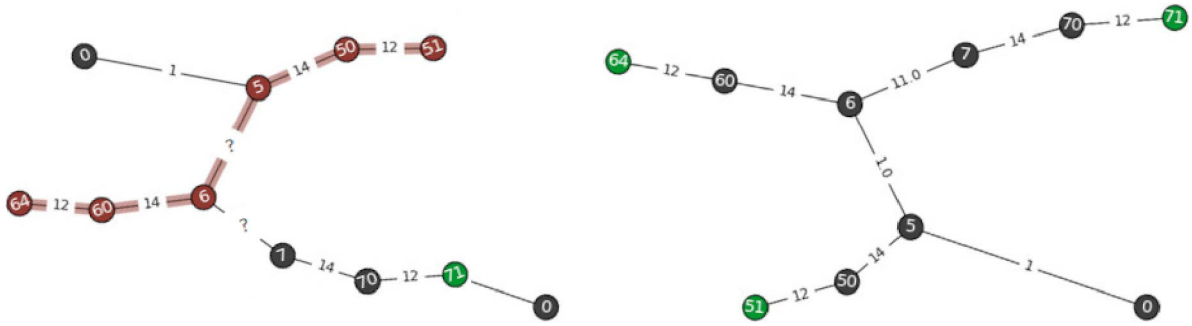
Figure 7.8: Matrix representation of the *MILP* allocation model.

The single constraints result in a two-dimensional matrix representation of the *MILP* model in Figure 7.8, whereby the total number of rows n is linearly related to the number of panels and edges of the scenario. The notation b_n corresponds to the right side of the equations. A *MILP* solver can then compute whether a solution to a particular objective function exists for all given constraints. Referring to the variable attenuators, the minimization of all attenuation values in G_A was set as the objective function.

MILP-Objective: *The sum of attenuation values for all panels $i \in P$ must be minimal.*

$$\textit{Minimize:} \sum_{i \in P} a_i$$

An example solution for the scenario in Figure 7.6 is given with the representation in Figure 7.9b. The allocation model and the visual graph generation is implemented in *Python* (part of the *ESF* project *pcap-forwarder*[3]) and is solved using *lpsolve*[4]. From a complexity-theoretic point of view, this problem is $\mathcal{NP}$-hard and this type of optimization problem is difficult to predict in terms of runtime. Initial practical tests showed that the runtime of the solvers increases dramatically, as the number of scenario nodes and links increases. The results of our analysis with increasing scenario constraints are presented and discussed in Section 7.4.



(a) Allocation path G_P for node 1 and 2 in G_S (b) Allocation graph G_A for the scenario G_S

Figure 7.9: Allocation of the scenario represented as attenuation graph G from connected panels. Note that all other nodes of a single panel (ref. Figure 7.7) are removed from G_H due to the allocation process.

[3] *pcap-forwarder*: `https://git.informatik.tu-cottbus.de/boehmse1/pcap-forwarder/`
[4] *lpsolve55*: `http://lpsolve.sourceforge.net/5.5/`

7.4 | Evaluation and Discussion

Our modeling strategy, the reference measurements, and allocation approach show the relevance especially of signal strength and attenuation value based emulation. Finally, we discuss the capabilities, enhancements, and limitations of the hardware-based channel emulation with respect to the *Split-Protocol-Stack* methodology.

7.4.1 | Feasibility of Hardware Allocation

Our prototype implementation of the hardware allocation approach does not serve for a full operational readiness, but it allow feasibility studies of scenarios with different node sizes which eventually results in different numbers of overall constraints for the *MILP* model. With the stepwise increase of nodes and edges in G_S, our analysis shows the principal feasibility of allocation using the *MILP* optimization on attenuation graphs. Due to the complexity of the model, however, there are constraints that are complex to solve and extremely increase the algorithmic runtime. Figure 7.10 below shows the runtime increase based on the iterations of the solver.

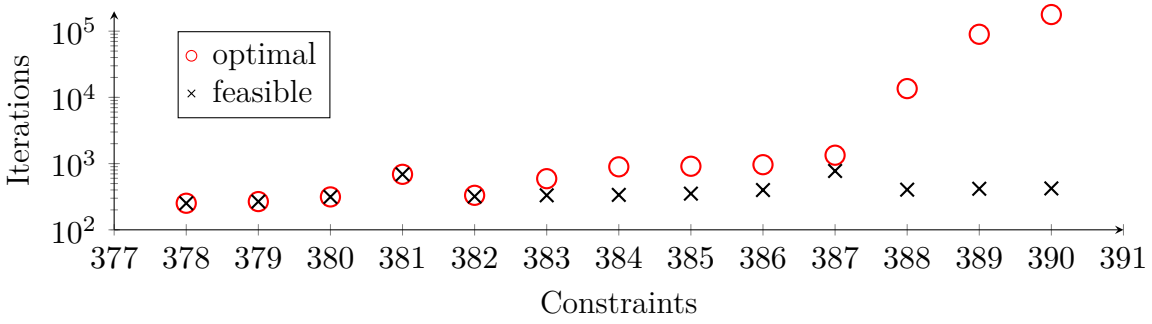


Figure 7.10: Number of iterations for solving the allocation model

The corresponding *MILP* model for the overall *RoSeNet* panel hardware architecture G_H incorporates 378 constraints modeled by 756 variables. Initial measurements of the allocation algorithm show a disproportionate increase in iterations and calculation time with the number of scenario constraints to get optimal solutions from the *MILP* solver (see Figure 7.10 – optimal). Adding only 12 additional scenario constraints (390 overall constraints), the solver needs 178213 iterations (approx. 20 s) to solve the model[5]. The first steps of the allocation process do not require optimal solutions, but only to show whether a model is feasible or not. To find a feasible solution the solver needs less than 1000 iterations independent of the number of constraints (see Figure 7.10 – feasible). Thus, while running on an Ubuntu Linux[5], the solver is able to calculate a feasible solution in less than 0.5 s.

[5] Ubuntu Linux (64 bit) virtual machine with 7 vCPUs (Intel Xeon X3470 Quad Core @ 2.93 GHz)

The developed solution approach currently assumes a static distribution of the nodes and relates to the subproblem of configuring the attenuation actuators. For the selection of nodes with respect to I_λ, different considerations can be taken into account (e.g., graph consolidation in [171]). Thus, the next step is to investigate, how this simplification can be removed. Besides fulfilling the algorithm precondition (ref. (1) in Algorithm 6), further indications for this are provided by the graph analysis, whereby the following parameters and dimensions have to be considered for attenuation graphs G:

- *Diameter* *(shortest path between each pair of vertices),*
- *Beta Index* *(ratio between the number of edges and nodes),*
- *Degree Centrality* *(number of edges that converge in a node),*
- *Closeness Centrality* *(indicates how close a node is to all others), and*
- *Betweenness Centrality* *(influence of nodes with a bridging function).*

In practice, the allocation problem can have more constraints relaying on node types, parameters and hardware inaccuracies. Our model provides a suitable basis for extension. Additional *MILP*-Constraints and multiple *MILP*-Objectives can easily be integrated into the model. Nevertheless, the practical applicability of the analyzed allocation and determination of attenuation values based on *MILP* must be considered in the application context. For example, live recalculation of G_A in scenarios with random mobility caused by mobility models of the simulation (ref. Subsection 2.2.5) must be excluded due to the long algorithmic runtime for optimal solutions. A deterministic pre-calculation, on the other hand, seems feasible, although at the same time very computational and time-intensive in advance of the emulation. In addition, the selection of nodes at the beginning of the allocation as well as a possible failure and recalculation in case of a negative result also affects the runtime and can readily lead to impossible allocations during the adaptation caused by node mobility. In our estimation, a mobility consideration is not practical with the current emulation hardware architecture, with respect to the extensions of the hardware attenuation discussed below, our model does not lose its importance at all.

7.4.2 | Enhancing the Hardware Attenuation

An analysis of the attenuation accuracy between individual nodes on adjacent panels according to the step attenuator configuration shows significant deviations of the target values (ref. Figure 7.11). This is due to the lack of *RF* shielding of the individual nodes, which allows signal power leakage out of the connectors. It can also be assumed that the *RF* transmission can be affected to a small extent by side effects, such as signal paths over solid wires on the panels. With respect to the measurements in the outdoor environment (ref. Section 7.2), these signal strength deviations are still comparatively small. However, additional shielding of the hardware nodes should be considered.

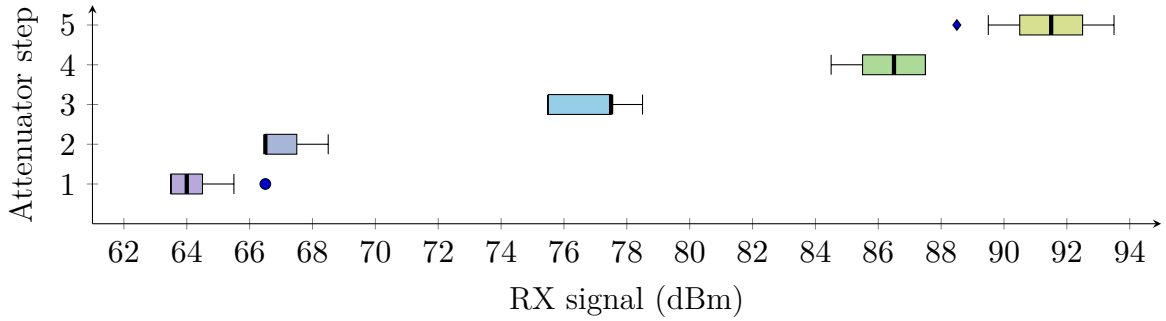


Figure 7.11: Attenuation measurements

Furthermore, the attenuation actuators at the panel input are not sufficient for a comprehensive channel emulation. Besides the results of the optimization which permits no real-time capability for live calculations, the variable attenuation between panels is generally problematic for node mobility because it is usually or at least for most scenarios not possible to *move* only single sensor nodes. An attenuation value change disturbs multiple nodes in a chain of the hardware graph G_H, but mobility can be approximated using node-based attenuation, either by adding attenuators for each node slot on the panel (at node c in Figure 7.7) or according to the *RIL PHY* modeling.

Moving the attenuation to the nodes or even providing a hardware emulation module for each slot of the panel hardware, so that even the nodes of a single panel in the chain can be shielded as desired, introduces fundamental changes to the panel-based hardware design. These extensions also add additional architecture-dependent attenuation to the system's *RF* path and to additional nodes $v \in H$ in G_H. Therefore, the scalability of the overall system decreases if these unavoidable effects cannot be compensated, e.g., with signal amplifiers. A discussion of the technical feasibility of this hardware design changes is beyond the scope of this thesis.

However, enabling node-based attenuation by configuring receiver sensitivity (`rxGain`) or/and transmit power (`txGain`) of individual nodes is suitable if provided by the particular node hardware. This must be included in the hardware attenuation graph G_H and the constraints of the *MILP* model based on the various node types $v \in N$. Thus, the hardware allocation process gets more fine-grained capabilities of the attenuation calculation and node mobility can be approximated or randomized, which is a commonly used approach for evaluation. Since especially the configuration of the receiver sensitivity (`rxGain`) is not supported by standard node hardware, the platform enhancement with *SDR RIL* modules is a comparatively simple upgrade to achieve. This extends the emulation capabilities enormously and makes, for example, the node-based *equalization* discussed in Subsection 7.1.2 available on the emulation platform while maintaining the current *RF* architecture.

8 | SEmulate Prototype and Use Case Study

SEmulate [54] is an abbreviation of the research project[1] and the prototype implementation of the evaluation concept, that seamlessly couples *S*imulation and *E*mulation. Seamlessly coupled means that the technology of hardware-based radio channel emulation is integrated into the network simulation without significantly changing the basic principle of running a *DES* scenario. *SEmulate* is intended for deployment as *Platform as a Service (PaaS)* for research and development projects of evaluating novel architectures and approaches in *WSNs* and the *IoT*.

In order to demonstrate the platform capabilities and the *Split-Protocol-Stack* emulation benefits, the analysis results, solution approaches, algorithms and components (the outcome of this thesis, ref. thesis outline in Figure 1.3, Chapters 4, 5, 6, and 7) must eventually be integrated into a prototype. Due to the massive dependencies in technologies, software, computational power, and (*RF*) hardware, we cannot present an evaluation system with full operational readiness at this time, but we can provide insights into the prototype architecture and first small-scale evaluation experiments. Regarding the reference configuration presented in Section 4.4, we demonstrate an evaluation scenario within the domain of cross-layer optimization strategies for wireless transmissions.

8.1 | SEmulate Overview

The way the different components are coupled follows exactly the basic methodology and central approach of this thesis as introduced in Section 4.2. How the conceptual and experimental achievements of the *Split-Protocol-Stack* approach are integrated in the prototype can be summarized as follows: The *Real-Time-Shift* synchronization and scheduling, presented in Section 5.2, is combined in the *PCAP ESF* as the core emulation runtime control module. The *PCAP* interface abstraction and modeling (Section 5.3) is implemented as central data communication container among the components. We have set up the nodes on the emulation hardware with our chip *RIL* implementation in *Contiki* (Section 6.2). The analysis of the wireless network planning (Subsection 7.1.1) is implemented in the prototype in the visualization and monitoring concept. The allocation of the *SEmulate* hardware platform resources, presented in Section 7.3, is implemented in the scenario generation process.

[1] The research project "Entwicklung eines Emulations- und Testsystems für robuste, drahtlose Sensornetze (SEmulate)" was funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) through the Central Innovation Programme (ZIM initiative) under Contract No. ZF4119201ED5.

8.1.1 | Emulation Prototype Architecture

SEmulate builds on the *RoSeNet* hardware architecture[2] and enables both a pure channel emulation and the coupling with a protocol simulator. The architecture in Figure 8.1 shows the coupling of the simulation and the emulation domain with the *PCAP* stream forwarder (*ESF*) in structure and methodology according to our basic definitions of *RIL* in Section 4.1. We now give a basic (but non-exhaustive) overview about the emulation *frontend* and *backend* realization.

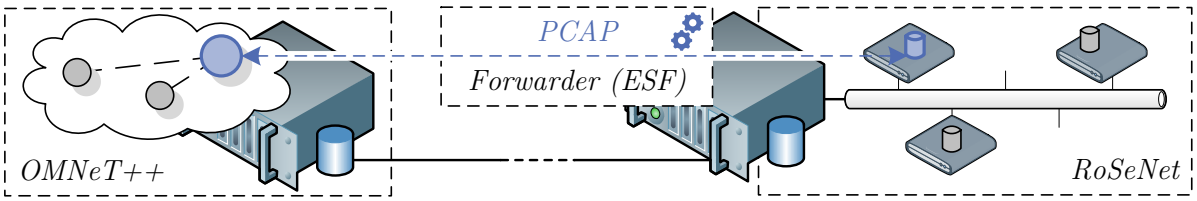


Figure 8.1: SEmulate Prototype *RIL* Architecture

The *SEmulate frontend*-design implements a multiple-client architecture for use in *administration*, pure *channel emulation*, and *Split-Protocol-Stack* parallel *simulation* and *emulation*. In the simulation domain, we have set up *OMNeT++* with our *Pseudo-Real-Time* scheduler and the customized models and interfaces for *IEEE 802.15.4*. All simulation scenario-related radio network and protocol modeling is based on *OMNeT++*/ INET. The *MAC* layer and the higher layers of the protocol stack are simulated, while the *PHY* and channel emulation domain is provided by the *SEmulate* control and *RoSeNet* hardware subsystem. The node positions for modeling the radio topology (ref. realization of a *UDG* in Section 6.1) and radio parameters are defined in the simulation *GUI* and translated into a representation for the emulation domain.

The emulation *backend* integrates a command and control module for interacting with the hardware components and for processing data to the chip *RIL* nodes. An overview of important control commands is given in the appendix Table C.1, where `cmdSendTransparentData` is used to transfer all *RIL*-related *PCAP* data to the nodes. We currently use nodes with ATmega128RFA1 radios[3] with our chip *RIL* implementation in *Contiki*, but the platform[2] can also be equipped with other node hardware and firmware. Furthermore, the interfacing to the simulation domain and the intermediate event scheduling are realized with the *PCAP*-based forwarder *ESF*. This component controls the time-accurate forwarding of events between the simulation and the *RIL* nodes while running a *Split-Protocol-Stack* emulation. The backend also provides a data base for all representation, configuration, and visualization for the pure emulation frontend.

[2] RoSeNet channel emulation and test platform for low-power wireless technologies, presented in Section 3.3.
[3] ATmega128RFA1: `https://www.microchip.com/ATmega128RFA1`

8.1.2 | Creating, Building, and Running Emulations

The scenario and parameter setup of a *Split-Protocol-Stack* emulation is inspired by the typical *OMNeT++* guidelines for the creation of simulation scenarios. Thus, the definition of the network topology including scenario, nodes, and protocol parameters are created based on `.ned` and `.ini` files in *OMNeT++*. They are included in the configuration at the *SEmulate* backend. The emulation domain also requires further hardware and *RF* configuration parameters as well as guidelines for the pure channel emulation. High-level specifications for the configuration of emulation scenarios address the following conceptual summary.

Static (manual), automatically assisted scenario configuration:

- *Node selection (node type, node count, identifiers),*
- *Node configuration (node platform, firmware, device parameters), and*
- *Physical links (radio topology).*

Static (automatic) scenario configuration:

- *Allocation (*RF *network configuration, attenuation, transceiver parameters),*
- *Node parameters (operating voltage, serial interface, digital/analog* IO*),*
- *Analysis functionality (packet sniffer, logging, energy/*IO *monitoring), and*
- *Firmware upload.*

Dynamic scenario configuration (manually for pure emulation mode, simulation-driven for the *Split-Protocol-Stack* approach):

- *Physical links (network join/leave, node mobility), and*
- *Node parameter (serial data, digital/analog* IOs*).*

Channel emulation and node-based configuration are classified into *static* and *dynamic*. The static scenario configuration mainly refers to the manual specifications of the radio network and hardware, including the hardware initialization and parameter setup. The dynamic configuration changes parameters at runtime (ref. hardware configuration commands in Table C.1, e.g., setting the attenuation values, switching on/off the power supply, or simulating *IOs* on the node input pins). As argued above, some configuration aspects are determined automatically, others must be created using suitable methods. We have made experimental investigations of the implementation for web-based configuration and visualization dashboards (ref. the prototype scenario configuration *Graphical User Interface (GUI)* in Figure C.2) for the pure emulation client independent of the *RIL* connection to the simulation. For monitoring, *SEmulate* enables connections to standard tools such as *Wireshark* and state-of-the-art data visualization techniques (e.g. for continuous monitoring of energy consumption).

Generation of the Emulation Configuration

The automated scenario generation and configuration of the *Split-Protocol-Stack RIL* emulation is essential for the practical usability of the approach. Our modeling using attenuation graphs provides a common basis for the hardware and scenario definitions and is tightly embedded in the backend of the prototype. Automatic resource allocation is one of the central algorithmic components of the model-based evaluation using *SEmulate*. In contrast, the creation of scenarios is based on the frameworks and file formats of the used simulation system. Whereas in *OMNeT++* the simulation program is executed using the parameters from the configuration files, in the *SEmulate* backend exactly these configuration files are retrieved to generate the respective hardware dependencies for the channel emulation.

Figure 8.2 shows, how file parsers, scripts, and configuration files are used to generate a *Split-Protocol-Stack* parallel simulation and emulation scenario. The so-called `NEDParser` translation script extracts the relevant definitions and parameters from an *OMNeT++* scenario and generates a new file format (`scenario.json`). The resource allocation script `NodeAllocator` eventually generates a hardware-dependent scenario by using the *RF* network configuration (`network.json`) and the deployed node types (`nodes.json`) of the emulation platform by means of the attenuation graph representation (ref. the main procedure calls of the allocation script in Listing C.13). Once an emulation scenario has been built, it can be executed with the configuration input parameters for the emulation domain (`network.json`, `scenario.json`) and the simulation domain (`omnetpp.ini`, `*.ned`). The emulation process then initializes the hardware and waits for *PCAP* packets from the simulation program. This setup and implementation seamlessly extends the process of building and running a simulation in *OMNeT++* (ref. Figure B.1). An excerpt from a scenario (`scenario.json`) generated from *OMNeT++* and allocated to the emulation hardware is shown in Listing C.12. Additionall, an excerpt from the initial *RF* network configuration is shown in Listing C.11.

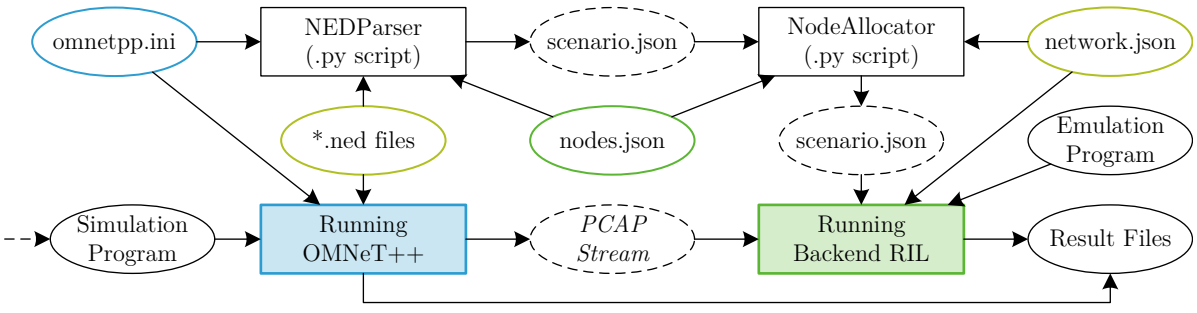


Figure 8.2: Overview of building and running an emulation with *SEmulate*

8.2 | Case Study Scenario: Cross-Layer Optimization

In contrast to modeling communications using reference designs (e.g., the *OSI* model in Figure 1.1a), protocols and architectures can be optimized by violating layered communication based on performance, management, or security constraints. This results in new interfaces, redefinitions of layer boundaries, or a common coordination of parameters among the layers [150]. However, implementing true node firmware for experimenting with cross-layer concepts is incredibly cumbersome. Network and protocol simulation are considered the state-of-the-art analysis and evaluation approach for cross-layer optimization due to its high flexibility in linking various modules and functions.

In this highly relevant research area, also a number of *IEEE 802.15.4*-based optimization approaches are reflected only in the purely simulative evaluation. Besides *MAC*-based optimization originating from higher protocol layers (e.g., *IEEE 802.15.4 Adaptive Access Parameter Tuning* in [182], or efficient *PAN* coordinator selection in [183] - both evaluated in pure *ns-2/ns-3*), a number of approaches that require high accuracy and close interaction of the *PHY* should also be mentioned. The *Transmission-Power Control (TPC)* approach in [184], evaluated in pure *ns-2/ns-3*, is based on location information tables of mobile networks. It defines new service primitives for passing node range information to the *PHY*. In contrast, *Receiver-Sensitivity Control (RSC)* in [185] is proposed to increase the energy efficiency of receiver-dominated nodes in *IoT* networks, evaluated based on pure virtual simulation in *OMNeT++*.

8.2.1 | Motivation and Objective

As mentioned above, optimization of performance parameters for transmitter and receiver is one of the main goals in cross-layer optimization regarding the *PHY* in *WSNs* and the *IoT* (e.g., increasing the energy efficiency with *Transmission-Power Control (TPC)* and *Receiver-Sensitivity Control (RSC)*). The *Split-Protocol-Stack* emulation can be used to incorporate accurate real-world *CSI* for energy-efficient and reliable *PHY* transmissions in network simulations. Accordingly, we create here a use case and reference evaluation scenario that serves as an example of a *pure wireless* configuration according to Subsection 4.4.2a / 4.4.3. It practically illustrates the contribution of parallel simulation and channel emulation for the evaluation of new approaches considering physical channel accuracy while maintaining protocol modeling flexibility. With this case study scenario, we demonstrate a receiver-dominated optimization based on the gain settings according to the radio packets signal strength measurement. In particular, it demonstrates the feasibility in a holistic *Split-Protocol-Stack* evaluation including *SDR RIL* transceivers. The overall goal of adjusting the receiver gain based on the *LQI* and *ED* using *SDR* is similar to a *feedback control* and aims at minimizing the energy consumption of a receiver.

Whereas the *ED* is the most important part of the channel selection algorithm, the *LQI* indicates the reliability of a used wireless link. It is often a decision criterion for whether a partial channel or route is selected for transmission (ref. *packet detection* in Subsection 2.2.2). For *IEEE 802.15.4* transceiver hardware, it is not specified, how to use the *LQI* value, but it shall be performed for each received radio packet. Mostly it is implemented using a *SNR* estimation and is represented as an integer ranging from 0 to 255 (lowest to highest quality). The result of the *ED* is also given in this range of values and represents the normalized received signal power (*RSSI*).

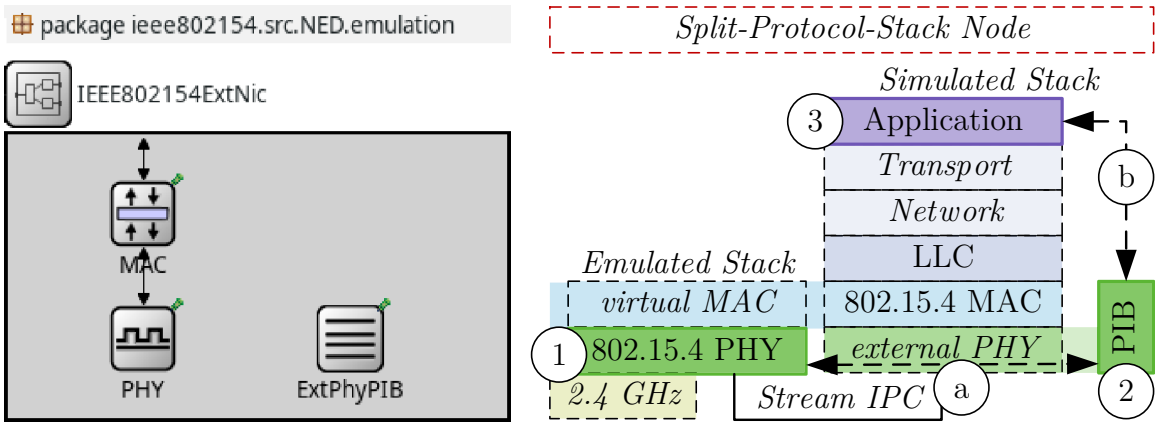
The variations of the *RSSI* in the reference measurements with deviations of several *dB* highlight the fact that environmental properties are hardly constant in outdoor applications (signal differences are 20 *dB* for a worst-case in Section 7.2). An efficient node placement with static configuration of the transmit power based on the signal strength measurements is therefore hardly feasible and wastes energy/battery capacity. Not least in the practical measurement studies, the potential for optimization became clear, as all packets are reliably received in a broader signal strength range. For example, with a bandwidth of approx. 20 *dB* difference in reception gain, all packets can be received without exception (ref. *RX* gain range 60 to 80 *dB* in Figure 6.9). The differences in the received signal strengths of the reference measurements also indicate highly reliable transmissions in the range of approx. 20 *dB* (ref. *RX* signal range −65 to −85 *dB* in Figure 7.5).

8.2.2 | Modeling and Definition

The abstract optimization goal for this scenario is to permanently keep the *LQI* for a communication link at 255 (maximum value) while minimizing the sensitivity at the receiver (*RX* gain). In existing *IEEE 802.15.4* chip implementations, the receiver sensitivity cannot be adjusted at all or at least not uniformly according to a common guideline. Moreover, the standard does not define any *SDU* and *PHY* parameters for setting the *RX* gain but only minimum values (ref. Table 2.1). In the context of the *RIL PHY*, introduced in Chapter 6, the modeling diversity of *SDR* transceivers is applied here. In a realistic channel emulation scenario, long-term measurements based on real transmissions can thus be evaluated without the need to develop real transceiver implementations for specific node hardware. An higher layer module in terms of an optimization algorithm in the simulation can be used to control this adjustments, but it requires access to the *PHY* and channel information. With only a few extensions, a simple information exchange across layer boundaries is enabled for parameterization of the *RIL PHY* transceiver hardware according to the *Split-Protocol-Stack*. Next, we explain these extensions to the simulation model.

PHY Information Database and Parameters

IEEE 802.15.4 [9, Sec. 6.4, pp. 45f] defines *PHY* constants which are hardware dependent (cannot be changed during operation) and *Personal Area Network Information Base (PIB)* attributes which are partially *read-only* parameters. As depicted in Figure 8.3, the *PIB* is provided in the simulation as a distributed database (simulation module) for all layers in this simplified cross-layer scenario. Therefore, new interfaces must be introduced in the involved protocol layers for communicating with this module (ref. ② in Figure 8.3b). To enable a higher layer to perform adjustments of the *RX* gain



(a) The *IEEE 802.15.4 OMNeT++* link layer module with external *PHY PIB*

(b) Cross-layer *PIB* interaction from a higher layer of the protocol stack

Figure 8.3: *IEEE 802.15.4* cross-layer communication based on an external *PIB*

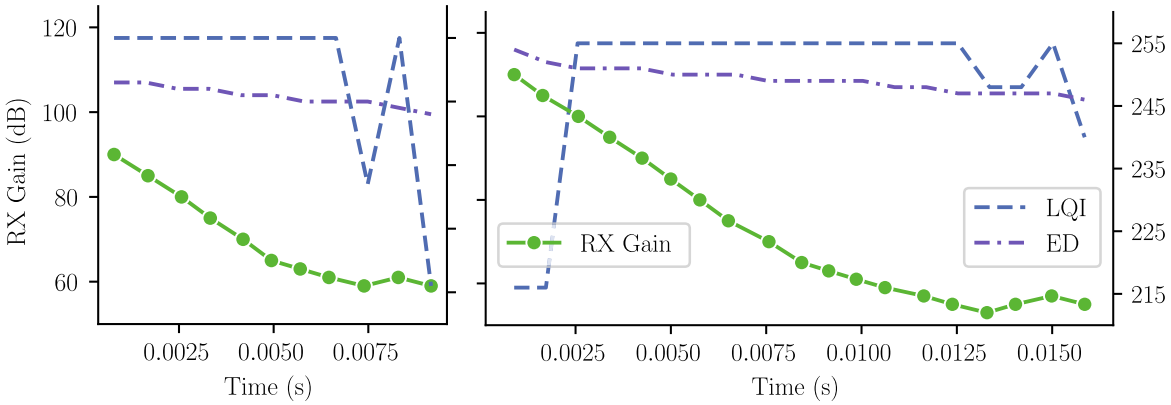
based on the current *LQI* and *ED* values additional parameters must be introduced. The additional hardware parameters are provided by extending the *PIB* defined in the standard with additional information. They are transmitted via the original protocol primitives (`PLME-SET.request`'s and `PLME-GET.request`'s). Furthermore, the resulting external *PIB* cross-layer database is updated according to parameter-individual requests or for all attributes simultaneously according to a new protocol primitive `PLME-GET-PHY-PIB`. The manipulation of the database on the *RIL* hardware is restricted to the *external PHY* simulation module (ref. communication ⓐ between ② and ① in Figure 8.3b), but higher layers can have access to the information and request to perform an attribute change (ⓑ). For the exemplary receiver-dominated cross-layer approach, the following additional *PIB* attributes are defined (ref. *PHY* parameters in Subsection 6.3.3).

- *phyLQI* *(link quality, generated from packet reception),*
- *phyEDvalue* *(channel energy, generated from signal strength),*
- *phyRXgain* *(sensitivity, according to the gain amplifier setup).*

8.2.3 | Results and Evaluation

The application layer module (③ in Figure 8.3b) is called `trafficgen` (adaptation of a standard *OMNeT++* module for traffic generation). It implements a simple feedback control process (ref. Listing C.6) according to the scenario definition and objective[4]. The scenario is started in a configuration in which radio packets are received with maximum channel quality ($phyLQI = 255$). A feedback follows after each received packet via a gradual decrease of the receiver's sensitivity by adjusting the corresponding *SDR* hardware parameters as long as the channel quality does not change. Once the *LQI* deviates downward, the sensitivity is increased again to compensate this deviation.

Using the prototype implementation in *SEmulate*, two measurement series with different radio channel settings were carried out as representative examples. A connection to the *RoSeNet* channel emulation could not be realized in this scenario, since the corresponding hardware interface connection for the *SDR RIL* concept are not implemented in the prototype. As consequence, uncontrollable real conditions arise for an indoor radio channel, which are also suitable for this case study and demonstration. In the first experimental setup, the transmitter and receiver were placed at a distance of 1 m from each other and an initial *RX* gain of 90 dBm was configured to avoid noise due to the analyzed overload with too high receiver sensitivity in Figure 6.9.



(a) *LQI*-based *RX* gain adaption (initialized with 90 *dB*)

(b) *LQI*- and *ED*-based *RX* gain adaption (initialized with 110 *dB*, *LQI* below maximum)

Figure 8.4: *RX* gain optimization from *LQI* and *ED* as a function of the simulation time[4]

The result in Figure 8.4a illustrates the potential of this type of receiver-dominated optimization. It shows the gradual reduction in gain at the receiver for each radio packet received over the simulated time. The *RX* gain at the antenna of the receiver can be reduced significantly (in the range of up to 30 *dB*) without degrading the channel quality

[4] The experimental measurement series were recorded in a supervised master's thesis [186].

and thus the fault tolerance for wireless data communication. Beyond a threshold of 59 *dB*, the channel quality *LQI* is no longer maximum under these conditions and is set back up one level by the feedback loop. Since the maximum value is again reached for the next radio packet, this process is now repeated continuously (depending on the conditions on the radio channel). Thus, this experimental setup demonstrates how to control a real-world wireless transmission based on an abstract algorithm within the network simulation.

The goal of the second experimental setup was to consider the largest possible link budget for the optimization. For this purpose, transmitter and receiver nodes were placed very close to each other (a few cm) and the measurement series were initially parameterized with a very high sensitivity of 110 *dB* gain at the receiver. Because of the high noise when dealing with high signal levels, the feedback algorithm additionally needs the signal energy *ED* to decide about reducing or increasing the *RX* gain (*LQI* below the maximum at high signal energy still leads to a reduction of *RX* gain). Thus, for each packet reception the *LQI* and *ED* values are evaluated for this exemplary demonstration. According to the results in Figure 8.4b, the link quality in this setup is maximum within a link budget of 45 *dB* (100 *dB* to 55 *dB*) which allows for significant energy consumption optimization on the receiver. The implementation of the event handling in the *OMNeT++* module `trafficgen` is presented in the appendix in Listing C.6.

8.2.4 | Discussion and Further Enhancements

The achieved results demonstrate the feasibility of the approach for modeling a cross-layer optimization strategy with the *Split-Protocol-Stack* emulation. For practical use, it shows the possibility to ensure reliable transmissions while reducing the gain in reception at the same time. In the same way, the transmitter can be parameterized to reduce the transmission energy in a control loop based on the evaluation of the received acknowledgments or to emulate the own node mobility. A corresponding attribute for the hardware configuration of the transmission power (*phyTXgain*) is also part of our extensions for the external *PIB*. Of course beyond this simple scenario, other criteria and parameters can be included in the optimization. Cross-layer consideration of parameters from several protocol layers is also conceivable, so that an intelligent algorithm considers both simulatively generated parameters and real hardware attributes from different node and network properties. Furthermore, if the *RIL* hardware modules are actually integrated and executed on the channel emulation platform the energy consumption of the nodes can be recorded simultaneously.

Some theoretical considerations or concrete extensions of the scenario may cover different state-of-the-art research areas. Regarding *security vulnerabilities*, the radio packet parameters can be evaluated in detail. If a packet is received with an untrustworthy, strongly deviating radio fingerprint (e.g., signal strength), an intelligent algorithm can trigger an

alarm or interrupt the communication. Regarding *dense radio conditions*, the transceiver can switch the configuration. If a source of interference generates strong ambient noise or technology interference within the channel emulation (see interference Figure 6.9), a transmitter algorithm can switch the transmission power or the actual channel and try to achieve error-free communication. Regarding *adaptive wireless transmissions*, the transceiver can monitor the channel activity. Artificial intelligence strategies evaluate several hardware parameters and *CSI* for a period of time to decide which physical parameters should be configured for a transmission. Modeling using the *Split-Protocol-Stack RIL* concept is perfectly suited for this kind of parameter studies from real-world measurements in the simulation. Existing intelligent or *ML* algorithms can easily be integrated in *OMNeT++* in a higher programming language (e.g., *C++* or *Python*).

9 | Conclusion

In this thesis, we proposed a novel approach for parallel simulation and emulation, called *Split-Protocol-Stack Wireless Network Emulation* with *RIL*. The incorporation of real radio hardware and physically accurately emulated radio channels into simulative studies has been a largely untouched area of research so far. Since the basic techniques partially involve diametrically opposed methodological concepts, the resulting research questions are quite challenging. For this purpose, theoretical concepts have been developed, prototypically implemented, and evaluated by means of concrete analyses, measurements, and scenarios to demonstrate the feasibility and benefits of a flexible, model-driven environment regarding accurate *MAC*, *PHY* and channel modeling.

9.1 | Achievements and Contributions

Link layer and radio channel modeling has been analyzed and discussed for simulation and emulation. From the discussed limitations of systems and approaches, the need for hybrid co-simulation and *HIL* has been derived. We have identified key requirements, current research questions, and problems which ultimately led into the conceptual architecture of our approach. With the *Split-Protocol-Stack*, we have proposed a methodology that seamlessly combines protocol simulation and hardware-based radio link emulation. Based on a detailed analysis of *HIL* and co-simulation techniques, we have conceptually compared our approach with state-of-the-art approaches. Furthermore, we have summarized the capabilities in the context of prototyping and evaluation, based on application domains, use cases, and reference configuration scenarios. In Chapter 4, key challenges and components of the *Split-Protocol-Stack* have been identified and analyzed, resulting in the following main issues: *synchronization* and *event scheduling*, radio transceiver *interfacing* and *abstraction*, and *scenario configuration* and *resource allocation*.

The *Real-Time-Shift* simulation in Chapter 5 introduces a *conservative* synchronization scheme which enables a *pseudo*-real-time event scheduling. We have contributed by extending the basic real-time *DES* concept with predetermined pause times for a certain scenario and system configuration. The *ESF* was introduced to enable the time-accurate event execution on real radio transceiver hardware from the protocol simulation. We have demonstrated that the scheduling accuracy does not significantly depend on the scenario configuration and resulting event stream. Furthermore, we provide the pseudo-real-time event scheduling approach with implementations for *OMNeT++* and the *ESF*.

RIL was introduced as the basic methodology, combining radio channel emulation with a *HIL* simulation concept in Chapter 6. We have contributed with two different implemenations of a *RIL PHY* transceiver on state-of-the-art radio hardware according to the *IEEE 802.15.4* reference protocol specification. *RIL* was realized as key abstraction component and interface to the radio channel which ensures that the commands initiated by the simulated *MAC* are executed accurately both in terms of radio parameters and timing. We have exemplarily demonstrated the *PHY* modeling for *RIL* simulations on transceiver chip and *SDR* hardware, have proved its feasibility, and have discussed the applicability of the two solutions. In particular, the *SDR RIL* enables accurate radio fingerprinting for future wireless network prototyping and evaluation.

In this thesis, relevant strategic details for scenario modeling and configuration have been summarized under the term *wireless network planning* in the context of *RIL* channel emulation in Chapter 7. According to the *resource allocation* of arbitrary scenarios, we have illustrated the complexity of node placement and pointed to limitations in the development of hardware-based emulation testbeds. We have demonstrated that an *MILP* optimization approach can solve the testbed configuration sub-problem in principle, but it is not sufficiently practical in this context on our exemplary emulation platform due to long run times. Based on our evaluation and discussion, we highlighted, however, which strategic enhancements for a *NET* and *RIL* modeling can mitigate this problem and what research questions arise from it.

The prototype *SEmulate* in Chapter 8 was initiated to develop a framework that implements relevant components, allowing for fundamental evaluation of the *Split-Protocol-Stack* approach. We have contributed with a respective model implementation for accurate *IEEE 802.15.4 RIL* transmissions. With a feasibility study according to a specified scenario from the *IEEE 802.15.4* specification, we have demonstrated the accuracy of our *RIL*-based simulation modeling. Moreover, the exemplary optimization strategy in our case study has experimentally shown, how the *Split-Protocol-Stack* approach using *SDR RIL* transceiver hardware enables the evaluation of a precise *PHY*-dependent research question. We have proven that the simulation is able to reliably perform radio hardware configurations and thus responds to varying conditions during real radio transmissions.

9.2 | Outlook

The *Split-Protocol-Stack* evaluation strategy is applicable to other protocols or systems and harmonizes the modeling process, since layers, interfaces and protocol messages are accurately modeled according to the standard specifications. They can be extended as needed for specific modeling purposes. Thus, the evaluation methodology enables cooperation and creates synergies in an interdisciplinary field of expertise. *Future work* will follow up on the key findings and prototypes from this thesis and incorporate them into existing workflows to provide evaluation tools for upcoming *research trends*.

Future Work

In addition to the contributions we have accomplished, there are further challenges to extend our basic concept and to be able to provide a powerful platform beyond the prototype implementation. Finally, we outline additional issues, practical challenges, and promising enhancements that have emerged throughout this thesis.

As demonstrated by our analyses in Chapter 7, the current architecture of the *RoSeNet RF* path does not provide the optimal conditions for an accurate representation of any scenario. Enhancing the hardware attenuation by including small-step controls between panels or even node-specific components on panels can significantly reduce the runtime of the optimization scheme by eliminating several constraints of the graph-based optimization model with a precise configuration option of the attenuators. These different approaches need to be further investigated, prototyped, and analyzed from the point of view of *RF*-technical feasibility, modeling benefits (radio topology, mobility), and impact on hardware allocation.

Moreover, accurate emulation of the node movement via signal fading on the current *RoSeNet RF* network must be excluded due to the issue that panel attenuator adjustments affect the attenuation on the entire chain. As demonstrated in the evaluation in Section 6.4 and with the cross-layer scenario in Section 8.2, *RIL PHY* modeling provides a well-suited prerequisite for node-individual, transmitter- and receiver-enabled fading emulation. However, this capability must be implemented tightly coupled to the hardware allocation. Position changes in the simulation model (*mobility modeling*) must be mapped to respective protocol services that change in the gain settings for sending and receiving on individual nodes. These are neither specified protocol message sequences nor protocol stack-related operations. In order not to corrupt simulation results configuration and time dependencies must be analyzed and specified according to an emulation run.

Implementing the *Real-Time-Shift* event stream processing (Section 5.2) on the *RoSeNet* panel hardware would be one of the most important further implementation steps. Since the *RIL* real-time stream processing in our prototype is currently implemented for a standard host system, accurate scheduling for the nodes on the channel emulation platform cannot be applied in practice up to now. Because the emulation controller on each panel is implemented on a market leading real-time operating system and is connected via a deterministic *LAN* to the emulation backend, the *PTP* is applicable for host synchronization. Furthermore, it should be investigated how *SDR* module integration and the full execution of *GNU Radio* can be implemented on the *RoSeNet* channel emulation hardware. In this context, minimizing the scheduling jitter of the event execution when using general-purpose *SDR* devices should be an important goal to reliably achieve maximum accuracy on the *RIL* hardware. We suggest to modify the static buffering of frames and provide *GNU Radio* with *Real-Time-Shift* scheduling at the hardware interface driver accordingly.

The presented *Real-Time-Shift* simulation introduced in Section 5.2 can theoretically be accelerated for a pure *Split-Protocol-Stack* node scenario. Because *virtual* events (e.g., higher-layer protocol messages) do not need to be scheduled in real-time, emulation runtime can be saved. This results in *hybrid* time scheduling, where purely virtual events run with virtual scheduling and *external* events cause a switch from virtual to real-time scheduling. Because the time compensation for processing emulated and external events on all involved components relies on the *GVT* and pre-defined time constants (τ_L introduced by the *RSV*), the *GVT* need to be additionally synchronized with the *ESF* before the emulated event is to be processed. How this proposal can be realized without compromising the accuracy of real-time scheduling remains to be investigated. By using the *PCAP*-based event exchange, for example, additional synchronization events can be integrated without changing the general event stream concept. However, this enhancement of running an *accelerated Real-Time-Shift* simulation can significantly speed up pure *Split-Protocol-Stack* emulation depending on the configuration in the application scenario (e.g., the *MAC BI*, protocol timeouts, or the time-controlled query of sensor values).

The full interaction with non-*RIL* wireless devices in a *Split-Protocol-Stack* evaluation setup requires pausing the nodes. Freezing a whole hardware circuit can theoretically achieved by stopping the internal oscillators or pausing the software routines by debugger interfaces (e.g., *JTAG*), but needs physical access to several hardware pins. This is possible on an *NET*, but leads to huge additional implementation efforts on the emulation hardware and backend. However, freezing the transceiver chip hardware during an active *TX* or *RX* operation causes proper wireless transmissions to fail. Interrupts can trigger an active wait within the program execution of the transceiver, but affect the regular program flow. Therefore, methods for synchronously pausing all attached real nodes need to be further investigated and analyzed.

According to the framework modeling philosophy (e.g., *OMNeT++*/INET modules, *ns-2/ns-3* model library, or *GNU Radio blocks*), the *Split-Protocol-Stack* emulation capabilities can be provided for various standards, protocols, or further wireless link layer technologies for which there is often already support for the used environments (e.g., *LoRaWAN* in *GNU Radio* and *OMNeT++*, cp. *FLoRa* in Section 2.3). For our purpose, it is planned to add further general parameters and different modulation schemes to the *SDR* hardware and *PHY* specifications according to the latest revisions of our reference protocol specification *IEEE 802.15.4*. Because due to the general abstraction in modeling for a certain specific purpose or application, a communication protocol or technology is often not represented with sufficient accuracy according to its specification and particular models cannot be reused for other purposes. Returning to the link layer technologies mentioned in Chapter 1, protocol messages and sequences are predominantly defined in their specifications. Thus, an important objective would be to specify several accuracy modeling guidelines for future research (discussed with the *Independent Verification and Validation (IV&V)* within the limitations of pure simulations in Section 2.4).

Research Trends

The battery life of wireless devices is certainly reduced by *RF pollution*, as the nodes have to transmit above the noise level which in turn creates additional interferences. If more interference occurs more bandwidth is required for information transmission because the specified data rate cannot be achieved on the channel. As digital transformation continues, area and hot-spot radio coverage is rapidly increasing through multiple technologies and is difficult to model for *radio network planning*. The current and upcoming *IoT* and *Smart City* concepts with countless surrounding wireless devices will make this situation much worse. *RF* pollution and interference are neither easy to determine nor predict, and even more difficult to model.

The purely simulative modeling of *RF* interference and pollution causes an incredible effort and therefore remains abstract or a theoretical consideration. However, taking these effects into account will become increasingly important in the prototyping and performance evaluation of modern techniques and optimization methods due to the expanding penetration of wireless technologies in the future. In our simplified case study optimization approach, we have already been able to demonstrate, how a realistic evaluation of intelligent approaches can be achieved using the *Split-Protocol-Stack*. Future research activities in wireless communication modeling will inevitably depend on the simulation of algorithmic flows with real-world accuracy or emulation capabilities.

Cross-layer optimization and *Cognitive Radio Sensor Networks (CRSNs)* have been a continuously growing, red-hot research focus for more than a decade now. In Section 8.2, cross-layer approaches to minimize energy consumption have already been presented with *Transmission-Power Control (TPC)* in [184] or *Receiver-Sensitivity Control (RSC)* in [185]. Current activities on *CRSNs* address, for example, research issues of *RF energy harvesting* for *spectrum-efficient networking* in the *IoT* [187] jointly consider radio transceiver design and network data transmissions. Concrete evaluation results are determined pure simulatively in each case. *PHY*- and channel-accurate model-based evaluation and prototyping of such approaches remains largely unclear.

Deep Learning for Future Wireless Communications is an emerging interdisciplinary paradigm which will revolutionize wireless networking by means of artificial intelligence. Recent studies, surveys, and approaches (e.g., [154, 156, 188]) show how *Deep Learning* on the physical layer and radio channel can stimulate transmission algorithms or transceiver design for coexistence of multiple radio access technologies or optimal transmission links for ultra-low latency constraints in dense networks. Specific solutions concern, for example, the channel coding in [189] or the modulation classification based on *Neural Networks* in [154]. Therefore, *radio fingerprinting* and *spectrum analysis* for feature engineering in *Neural Networks* are key elements. We consider our *SDR RIL* methodology for acquiring features of the radio channel to enable prototyping for reconfigurable *PHYs* using software building blocks an important step in this direction.

A | Analysis and Discussion Details

Within the scope of this thesis, an analytical discussion and comparison of selected approaches and systems is provided. To summarize the results of this analysis, a visual representation and overview is given in the analytical part using radar plots. Note that the evaluation scale does not represent defined quantities, but a linear level of characteristics or performance from very poor (center of the radar plot) to excellent. Points are awarded based on actual system parameters and after comparing or ranking the approaches against each other. Subsequently, the details of this comparative analysis of systems and approaches are given in addition to the respective visualization.

A.1 | Analysis of Radio-in-the-Loop Solutions

	Chip Radio-in-the-Loop	*Software Radio-in-the-Loop*
Performance	(3) sufficient for most scenarios	(5) beyond most specifications
Flexibility	(4) replaceable HW, node operating system config	(5) configurable HW parameters and transceiver SW
Configurability	(3) *PHY* attributes/parameters according to the platform	(5) all *PHY* attributes and parameters
Affordability	(5) low cost transceivers possible	(3) usually expensive
Accuracy	(6) testbed-like high accuracy	(4) mid-high (not adapted to attributes in specification)

Performance - (1) insufficient, (2) bad, (3) sufficient, (4) good, (5) very good, (6) unlimited
Flexibility - (1) no, (2) replace SW, (3) SW conf, (4) replace HW, (5) HW/SW conf, (6) adjust HW
Configurability - (1) no, (2) PHY attributes, (3) PHY params, (4) switch PHY, (5) adjust PHY, (6) all
Affordability - (1) cost-blasting, (2) very expensive, (3) expensive, (4) affordable, (5) low, (6) very low
Accuracy - (1) unusable, (2) low-mid, (3) mid, (4) mid-high, (5) high, (6) testbed-like

Table A.1: Analysis of Radio-in-the-Loop solutions, according to Figure 6.13

A.2 | Analysis of Radio Link Emulation Systems

	Jung et al. [111]	Ludwig et al. [115]	Beshay et al. [112]	Beuran et al. [113]	Elsner et al. [114]
Scalability	(1) exact setup with 4 nodes	(5) 1000 nodes	(4) potentially hundreds	(6) large-scale (beyond 1000)	(1) exact setup with 4 nodes
Mobility	(4) attenuation-based	(4) attenuation-based	(5) mobility patterns	(5) mobility patterns	(3) potentially, not defined
Modularity	(2) individual testbed	(5) modular HW installation, framework	(5) HW framework	(6) testbed framework	(5) HW/SW framework
Configurability	(1) manual setup	(5) emulation control backend	(4) SW with control *GUI*	(5) testbed control backend	(4) SW with control *GUI*
Traceability	(1) no automated tracing	(5) full hardware and *RF* monitoring	(4) status monitoring	(4) testbed node monitoring	(4) SW node monitoring
Accuracy	(6) real devices, exact setup	(5) real devices, emulated *RF*	(4) real devices, channel sim	(2) model-based, abstracted	(4) propagation simulation
Isolation	(5) coax connected and shielded	(4) coax connected	(4) coax connected	(5) disturbed on testbed *LAN*	(6) full virtual

Properties scale - (1) fixed/no/low, (2) low-mid, (3) mid, (4) mid-high, (5) high, (6) very high/full/arbitrary

Table A.2: Analysis of radio link emulation systems, according to Figure 3.3

A.3 | Analysis of HIL and Co-Simulation Approaches

	Zhang et al. [142]	Weingärtner et al. [140]	Unterschütz et al. [141]	Wehner et al. [143]
Scalability	(4) hundreds of nodes, but time increases when using co-sim nodes	(6) theoretically unlimited (experiments with 15,000)	(4) approx. 100 nodes	(2) limited to the gateway scenario
Mobility	(3) not specified, potentially with *TOSSIM* channel model	(5) not specified, potentially pure virtual in *ns-2/ns-3*	(5) simulation model-based virtual mobility	(2) manual, field test
Configurability	(3) only *TinyOS*, real code/no model based adjustments	(4) modular for link layer, specific for host protocol stack	(4) only for *CometOS* implementations	(3) propreitary fixed, generic simulation interfaces
Traceability	(5) simlation/emulation events, not for higher layers	(4) simulation events for *MAC/PHY*/channel, stack/packet trace, not for higher layers	(4) simulation events, not for real nodes	(2) simulation events for communication link, not for real nodes
Accuracy	(5) real code, transceiver chip & channel model	(4) real protocols, model based channel	(5) real code	(3) real hardware and code, no channel model

Scalability - (1) no, (2) few nodes, (3) dozens, (4) hundreds, (5) thousands, (6) unlimited
Mobility - (1) no, (2) virtual easy, (3) virtual enhanced, (4) virtual complex, (5) hw easy, (6) hw enhanced
Configurability - (1) no, (2) simu param, (3) simu models, (4) emu param, (5) emu models, (6) model independent
Traceability - (1) no, (2) simu specific, (3) simu model, (4) simu insight, (5) emu packet trace, (6) simu & emu stack
Accuracy - (1) low, (2) low-mid, (3) mid, (4) mid-high, (5) high, (6) testbed-like

Table A.3: Analysis of *HIL* and co-simulation approaches, according to Figure 4.3

	Mittag et al. [52]	Obermaier et al. [145]	Ding et al. [124]	*Split-Protocol-Stack*
Scalability	(4) hundreds of nodes	(1) single gateway node	(2) not measured (low scale tests with 3 nodes)	(5) *1000, e.g., according to RoSeNet* hardware
Mobility	(2) not specified, potentially based on the channel model	(3) mobility model for sim, no for real	(5) analog but only attenuation based	(5) *attenuation-based, simulation-driven*
Configurability	(4) modular but specific	(3) specific in terms of simulation models and real communication	(3) modular but specific to radio link, unknown for protocols	(5) *modular emulation control, simulation models*
Traceability	(5) simulation events, *PHY*/channel performance parameters	(4) simulation events	(3) specific performance parameters	(6) *simulation events, node hardware, and RF channel*
Accuracy	(5) *PHY* modulation simulation	(5) attenuation-based emulated radio link, higher-layer protocol simulation	(4) radio interface and channel, no higher-layer protocol simulation	(5) *attenuation-based emulated radio link, higher-layer protocol simulation*

Scalability - (1) no, (2) few nodes, (3) dozens, (4) hundreds, (5) thousands, (6) unlimited
Mobility - (1) no, (2) virtual easy, (3) virtual enhanced, (4) virtual complex, (5) hw easy, (6) hw enhanced
Configurability - (1) no, (2) simu param, (3) simu models, (4) emu param, (5) emu models, (6) model independent
Traceability - (1) no, (2) simu specific, (3) simu model, (4) simu insight, (5) emu packet trace, (6) simu & emu stack
Accuracy - (1) low, (2) low-mid, (3) mid, (4) mid-high, (5) high, (6) testbed-like

Table A.4: Analysis of *HIL* and co-simulation approaches, according to Figure 4.4

B | OMNeT++ / INET Overview

OMNeT++ [48] (Objective Modular Network Testbed in C++) is basically a generic simulation framework for research and development of complex distributed systems with a rapidly growing scientific community and a long period of active development. Many extensions, frameworks, and simulation models became available, most of them open-source, particularly for communication network areas, e.g., internet protocols, wireless networks, *WSNs*, mobile ad-hoc and mesh networks, or vehicular networks. On of the largest model framework in this context is INET, which provides communication protocols, network devices, algorithmic models, and data structures.

The C++ *DES* kernel provides support and gives a structure to model simulation components following the event scheduling approach. It permits to parametrize models, add and control random numbers, collect statistical results, support graphics and animations *GUI*, and much more. Unlike many other network simulators with often fixed software, *OMNeT++* features a generic component architecture that allows the model builder to decide how network nodes, interfaces, protocols, and applications are aligned to form an overall simulation model component. This also allows components to be used as libraries or framework for different scenarios.

B.1 | Simulation Architecture and Components

The component model of *OMNeT++* (cp. [48, Sec. 1.3]) consists of four key components: *simple* and *compound modules*, *parameters*, and *connections*. A dynamic simulation model consists of active C++ modules that communicate with each other by passing messages. In network simulation, simple modules are, for example, traffic sources/sinks, protocol entities, routing tables, or parameter control algorithms, while compound modules are assembled from simple modules, e.g., host nodes or network routers (ref. Figure 2.7a in Subsection 2.3.2). Modules can have *parameters*, initialized with default values, to provide configuration data for a unique simulation run. This parameters that may also be random to pass stochastic input to modules can be defined for all modules in one single configuration file (`omnetpp.ini`) for reproducible simulation runs.

Messages are exchanged between modules over their *connections*, and they may contain arbitrary data in addition to predefined attributes such as timestamps. A small and compact message definition language (`*.msg`) is used to define custom message types that can consider, for instance, header parameters or any payload data like *PDUs* in

communication protocols. The modules are equipped with *gates* for message input and output to enable a connection to another module. Because of the hierarchical structuring, messages typically travel through a chain of connections, where properties, such as *propagation delay*, *data rate*, and *bit error rate* can be assigned each. To specify the modules and their connections in a topology of the entire simulation network, *OMNeT++* uses a separate network description language (`*.ned`).

B.2 | Creating, Building, and Running Simulations

For the modeling of wireless communication systems, for example, the INET framework provides a multitude of simulation models for upper layer protocols and of models for nodes, devices, applications, and many more (cp. [48, Ch. 2] for a detailed overview of the framework ecosystem). When creating a simulation, the user has to edit the network definition that models the topology of the network and to parametrize the simulation with configuration parameters. For example, also the scheduler class can be defined in the simulation configuration file, which is fundamental different when running real-time *DES* (ref. Section 2.1.3). In [48, Fig. 2.2, p. 60] there is an example of how components of a communication system, especially the protocols and applications, are connected to each other.

The software for *OMNeT++* simulations is compiled and built using a `Makefile` which lists the dependencies between all files (e.g., modules and message definitions), and instructions about simulation libraries to include in the simulation program. Once a simulation has been built, it can be executed by taking the configuration parameters (`omnetpp.ini`) and network definitions (`*.ned`) for setting up the simulation run (cp. [48, Ch. 1] for further reading). A detailed overview about the module development of and the process of assembling and and running simulations, as well as the *OMNeT++* ecosystem can be found in [48]. In Figure B.1 a basic overview of building and running a simulation in *OMNeT++* is shown.

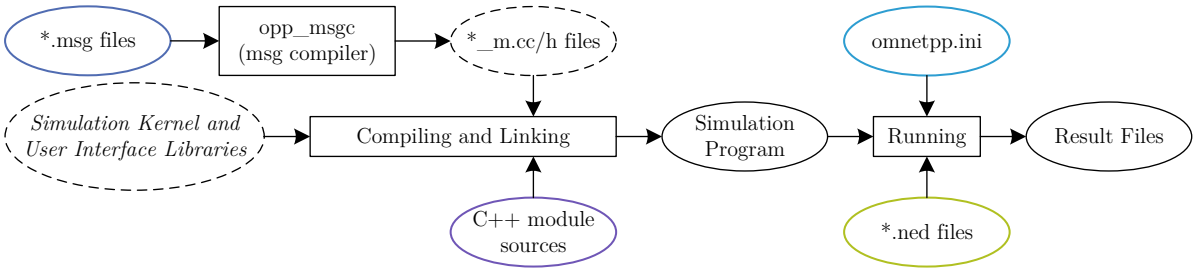


Figure B.1: Overview of building and running a simulation in *OMNeT++*

C | Implementation and Configuration Details

Subsequently, selected implementation and configuration details are highlighted according to major components of the *SEmulate* prototype and the evaluation levels of the *Split-Protocol-Stack* using short excerpts. The complete source code of the software components, as well as configuration files, measurement results, and automation scripts are provided with the repositories as specified.

C.1 | OMNeT++

```
/** init buffer from pcap block message event */
EPB *epb = check_and_cast<EPB *>(msg);
Buffer b(pcapBlock, epb->getDataArraySize());

/** deserialize sdu header and encapsulated pdu */
psdu *sdu = check_and_cast<psdu *>((IEEE802154Serializer().deserializeSDU(b)));
mpdu *pdu = check_and_cast<mpdu *>((IEEE802154Serializer().deserialize(b)));
sdu->encapsulate(pdu);

/** get corresponding node's virtual phy submodule and send to it */
cModule *mod = simulation.getModule(interfaceTable[epb->getInterface()]);
cModule *phy = mod->getSubmodule("NIC")->getSubmodule("PHY");
sendDirect(sdu, phy, "inFromExt");
```

Listing C.1: Exemplary deserialization of a *PD DataIndication* from an incoming external *EPB* in *OMNeT++*

```
85 void IEEE802154ExtPhy::handleMessage(cMessage *msg)
86 {
87     if (msg->arrivedOn("PLME_SAP")) { // --> Message arrived from MAC over PLME
88         switch (mappedUpperLayerMsgTypes[msg->getName()]) // --> Management Requests
89         {
90             case SETTRXSTATE: {
91                 // instruct external PHY to set the TRX state
92                 sendDirect(msg, simulation.getModule(extInterfaceID), "inDirect");
93                 break;
94             }
```

Listing C.2: External PHY event handling - excerpt from `IEEE802154ExtPhy.cc`

The following listings C.3 and C.4 depict details of our *Real-Time-Shift* enhancements, introducing new core event types *pause* and *resume*.

```
340     if (msg->getKind()==msgKindPauseEvent) {
341         // let all modules wait for delay timeval
342         delayScheduledMessages(SimTime(delayEventsMs, SIMTIME_MS));
343         paused = true;
344     } else if (msg->getKind()==msgKindResumeEvent) {
345         // shift the simulation time back and reset the events
346         sim->setSimTime(sim->getSimTime()-SimTime(delayEventsMs, SIMTIME_MS));
347         // reset the event delay
348         resetDelayedMessages();
349         paused = false;
350     }
```

Listing C.3: *Real-Time-Shift* synchronization event scheduling - excerpt from `*getNextEvent()` in `RealTimeShiftScheduler.cc`

```
94  void RealTimeShiftScheduler::delayScheduledMessages(simtime_t time)
95  {
96      // add time offset for all events except pause/resume and remember in
            delayMessages
97      for (cMessageHeap::Iterator it(sim->msgQueue); !it.end(); it++) {
98          cMessage *msg = it();
99          if(msg->getKind()!=msgKindPauseEvent && msg->getKind()!=msgKindResumeEvent) {
100             if (msg->getArrivalTime() < sim->getSimTime() + time) {
101                 // remember arrival time
102                 msg->setTimestamp(msg->getArrivalTime());
103                 msg->setArrival(msg->getArrivalModule(), msg->getArrivalGateId(), msg->
                        getArrivalTime() + time);
104                 delayMessages.insert({msg->getId(), msg});
105             }
106         }
107     }
108 }
109
110 void RealTimeShiftScheduler::resetDelayedMessages()
111 {
112     for (auto it=delayMessages.begin(); it != delayMessages.end(); ++it) {
113         cMessage *msg = it->second;
114         msg->setArrival(msg->getArrivalModule(), msg->getArrivalGateId(), msg->
                getTimestamp());
115     }
116     delayMessages.clear();
117 }
```

Listing C.4: *Real-Time-Shift* scheduler event shift procedures - excerpt from `RealTimeShiftScheduler.cc`

C.1.1 | Scenario Configuration

As an example reference scenario for the feasibility studies, we choose the *PAN* start based on an active channel scan [9, pp. 217ff]. Furthermore, various configuration and scenario examples are provided with the *IEEE 802.15.4* model implementation [96][1].

```
[General]

scheduler-class = "RealTimeShiftScheduler"

*.IEEE802154Nodes[*].Network.stdLLC.PANId = 7
*.IEEE802154Nodes[*].NIC.MAC.IEEE802154Mac.defChann = 25
*.IEEE802154Nodes[*].NIC.MAC.IEEE802154Mac.isFFD = true
*.IEEE802154Nodes[*].NIC.PHY.CCAMode = 1
*.IEEE802154Nodes[*].NIC.PHY.transmitPower = 1
*.IEEE802154Nodes[*].NIC.PHY.currentChannel = 25    # 2475 MHz
*.IEEE802154Nodes[*].mobilityType = "StationaryMobility" # no mobility model

network = ieee802154.simulations.net

###--------- Scenario: Single FFD starting a WPAN ---------###
# One single and static IEEE 802.15.4 Host
# Single host is starting on the communication channel,
# searches for other nodes and starts a WPAN
# Scenario to test out correct WPAN starting procedure
# CAP transmission for beacon-enabled PAN
# Beacon Order    = 6 -> Beacon Interval  61440 Symbols = 983.04ms
# Superframe Order = 6 -> Superframe Duration 61440 Symbols = 983.04ms
# Duty Cycle      = 100.00 %
###-----------------------------------------------------------------###
[Config StartWPAN-1Node_Starting_WPAN]

sim-time-limit = 10s    # enough time to run through the startup process
*.numHosts=1

*.IEEE802154Nodes[0].application.protocol = 1
*.IEEE802154Nodes[0].application.sendInterval = 1s
*.IEEE802154Nodes[0].mobility.initialX = 200m
*.IEEE802154Nodes[0].mobility.initialY = 200m
*.IEEE802154Nodes[0].mobility.initialZ = 0m
```

Listing C.5: *OMNeT++* scenario config - excerpt from `omnetpp.ini/scenarios.ini`

[1] *IEEE802154*: `https://git.informatik.tu-cottbus.de/boehmse1/IEEE802154INET-Standalone`

C.1.2 | Cross-layer Optimization Scenario

```
138 void trafficgen::handleMessage(cMessage *msg)
139 {
140    if (mappedMsgTypes[msg->getName()]==GETCONFPPIB) {
141
142        GetPPIBConfirm *conf=check_and_cast<GetPPIBConfirm*>(msg);
143        lqi=conf->getPIBLQI();
144        rxGain=conf->getPIBrxgain();
145        ED=conf->getPIBsignalstrength();
146
147        lqihist.collect(lqi);
148        lqivec.record(lqi);
149        EDhist.collect(ED);
150        EDvec.record(ED);
151        rxgainhist.collect(rxGain);
152        rxgainvec.record(rxGain);
153
154        if(conf->getPIBLQI()==255) {
155            std::cout<<"RX gain = "<< rxGain<<std::endl;
156            SetRequest* PhyPIBSet=new SetRequest("PLME-SET.request");
157            PhyPIBSet->setPIBattr(rxgain);
158
159            if(rxGain<=51) PhyPIBSet->setValue(rxGain-1);
160            else if(rxGain<=65) PhyPIBSet->setValue(rxGain-2);
161            else if(rxGain<=115) PhyPIBSet->setValue(rxGain-5);
162
163            sendDirect(PhyPIBSet,simulation.getModule(ppibID), "inDirect");
164
165        }
166        if(lqi<255){
167            SetRequest* PhyPIBSet=new SetRequest("PLME-SET.request");
168            PhyPIBSet->setPIBattr(rxgain);
169
170            if(ED<252){
171                if(rxGain<=51) PhyPIBSet->setValue(rxGain+1);
172                else if(rxGain<=65) PhyPIBSet->setValue(rxGain+2);
173                else if(rxGain<=115) PhyPIBSet->setValue(rxGain+5);
174            } else {
175                if(rxGain<=51) PhyPIBSet->setValue(rxGain-1);
176                else if(rxGain<=65) PhyPIBSet->setValue(rxGain-2);
177                else if(rxGain<=115) PhyPIBSet->setValue(rxGain-5);
178            }
179
180            sendDirect(PhyPIBSet,simulation.getModule(ppibID), "inDirect");
181        }
```

Listing C.6: Cross-layer optimization event handling - excerpt from `trafficgen.cc`

C.2 | SEmulate Backend

C.2.1 | Radio-in-the-Loop Event Stream Forwarder

In contrast to the scheduler in *OMNeT++*, the *ESF* scheduling processes the emulated events to the corresponding real-world radio hardware, according to the shifted high resolution timestamp.

```
398 void TimedTerminalForwarder::getNextEvent() {
399     std::chrono::time_point<std::chrono::high_resolution_clock> ts_now;
400 
401     while (this->run) {
402         if (sim_start) {
403             // dequeue element
404             auto next = this->queue.dequeue();
405 
406             if ((next.type != EVENT_SHIFT) && (next.type != EVENT_EMULATED)) {
407                 throw std::runtime_error("Invalid Event Type in FES: " + std::string(
                        eventTypeToString(next.type)));
408             }
409 
410             // get simulation time
411             ts_now = clock.simTime();
412 
413             logEventProcessing(ts_now, &next, EVENT_LOG_DEQUEUE);
414 
415             // wait until time arrives
416             if (next.time > ts_now) {
417                 if (waitUntil(next.time)){
418                     // mutex locked time accurate event processing
419                     processEvent(ts_now, &next, true);
420                     logEventProcessing(ts_now, &next, EVENT_LOG_PROCESS);
421                 }
422                 else
423                     Logger::moduleLog(dmodule, "ERROR: waitUntil.");
424             } else {
425                 processEvent(ts_now, &next, true);
426                 logEventProcessing(ts_now, &next, EVENT_LOG_PROCESS);
427                 Logger::moduleLog(dmodule, "WARNING: time is behind execution!");
428             }
429 
430             delete[](next.pcap_data);
431         }
432     }
433 }
```

Listing C.7: *Event Stream Forwarder* scheduler thread main routine - excerpt from `TimedTerminalForwarder.cpp`

C.2.2 | Radio-in-the-Loop Transceiver Implementation

```
252 PROCESS_THREAD(transceiver_init, ev, data)
253 {
254     static pcapng_enhanced_packet_block_s packet;
255     static PHY_msg msg;
256     static uint8_t packetCounter = 0;
257 
258     PROCESS_BEGIN();
259 
260     /* init module */
261     initialize();
262 
263     /* wait for packet input */
264     while (1) {
265         packetCounter++;
266         memset(&packet, 0, sizeof(packet));
267         memset(&msg, 0, sizeof(msg));
268         PROCESS_WAIT_EVENT_UNTIL(ev == pcapng_event_epb);
269         pcapng_line_read_epb((uint8_t *)data, &packet);
270         /* deserialize and handle encapsulated message */
271         deserialize_msg((uint8_t *)data + sizeof(pcapng_block_header_s) + sizeof(
                pcapng_enhanced_packet_block_s), &msg);
272         handleMessage(&msg);
273     }
274 
275     PROCESS_END();
276 }
```

Listing C.8: Nodes transceiver process in Contiki - excerpt from `transceiver.c`

```
#define NETSTACK_CONF_NETWORK     nullnet_driver
#define NETSTACK_CONF_MAC         nullmac_driver
#define NETSTACK_CONF_RDC         transceiver_rdc_driver
#define NETSTACK_CONF_RADIO       rf230_driver
#define CHANNEL_802_15_4          26

#define RF230_MAX_TX_POWER        15
#define RF230_MIN_RX_POWER        30
#define RF230_MAX_ED_THRESHOLD    15
#define RF230_CONF_AUTOACK        0
#define RF230_CONF_RX_BUFFERS     3
```

Listing C.9: Netstack and driver configuration in Contiki - excerpt from `contiki-conf.h`

```
296 void *transceiver_main_thread(void *data)
297 {
298     static pcapng_enhanced_packet_block_s epb;
299     static PHY_msg phy_msg;
300     static uint8_t packetCounter = 0;
301 
302     while (1) {
303         msg_t msg;
304         uart_event_msg_t *block_event;
305 
306         /** wait for packet input */
307         msg_receive(&msg);
308         block_event = (uart_event_msg_t *)msg.content.ptr;
309         packetCounter++;
310 
311         switch (block_event->type) {
312             case PCAPNG_BLOCK_TYPE_EPB:
313                 memset(&epb, 0, sizeof(epb));
314                 memset(&phy_msg, 0, sizeof(phy_msg));
315                 uart_pcap_read_epb(block_event->data, &epb);
316                 /* deserialize and handle encapsulated message */
317                 deserialize_msg(block_event->data + sizeof(pcapng_block_header_s) +
                        sizeof(pcapng_enhanced_packet_block_s), &phy_msg);
318                 handleMessage(&phy_msg);
319                 break;
320         }
321     }
322 
323     return NULL;
324 }
325 
326 int main(void)
327 {
328     puts("IEEE 802.15.4 Serial PHY Transceiver.");
329 
330     /** create transceiver main thread */
331     transceiver_pid = thread_create(transceiver_thread_stack, sizeof(
            transceiver_thread_stack), THREAD_PRIORITY_MAIN - 2, THREAD_CREATE_SLEEPING,
            transceiver_main_thread, NULL, "transceiver");
332 
333     /** initialize uart pcap interface */
334     interface_init();
335 
336     /** start main thread */
337     thread_wakeup(transceiver_pid);
338 
339     return 0;
340 }
```

Listing C.10: Nodes transceiver process in RIOT - excerpt from `main.c`

C.2.3 | Prototype Configuration

```
{
  "panels": [
    {
      "name": "Pnl-5",
      "IPAddress": "192.168.0.245",
      "MACId": "11.22.33.44.55.66",
      "id": 500,
      "chainNumber": 0,
      "chainPos": 0,
      "nodes": [
        {
          "name": "Node-0",
          "id": 501,
          "slot": 0,
          "type": "RCB128RFA1"
        },
        {
          "name": "Node-1",
          "id": 502,
          "slot": 1,
          "type": "RCB128RFA1"
        },
        {
          "name": "Node-2",
          "id": 503,
          "slot": 2,
          "type": "RCB128RFA1"
        },
        {
          "name": "Node-3",
          "id": 504,
          "slot": 3,
          "type": "RCB128RFA1"
```

Listing (C.11) *SEmulate RF* network - excerpt from `network.json`

```
{
  "radio": {
    "protocol": "802154",
    "hfband": 2400,
    "channel": 24
  },
  "devices": {
    "panels": [
      {
        "id": 500,
        "attenuation": 1.0
      },
      {
        "id": 600,
        "attenuation": 11.0
      },
      {
        "id": 700,
        "attenuation": 1.0
      }
    ],
    "nodes": [
      {
        "id": 501,
        "type": "RCB128RFA1",
        "range": 50,
        "y": 48.5,
        "x": 21,
        "serial": {
          "baud": "57600",
          "parameter": "8N1"
        },
        "firmware": "transceiver.hex",
```

Listing (C.12) *SEmulate* scenario allocation - excerpt from `scenario.json`

```
from AttenuationGraph import NetworkAttenuationGraph, ScenarioAttenuationGraph,
    ScenarioAllocationGraph

scenario = ScenarioAttenuationGraph('scenario.json')
network = NetworkAttenuationGraph('network.json', 'nodes.json')
allocation = ScenarioAllocationGraph(network, scenario)
```

Listing C.13: Exemplary main *Python* script for generating the allocation based on scenario and *RF* network configuration files - `allocateNodes.py`

	Description
`cmdSetVcc`	Sets the supply voltage for the radio nodes of a panel.
`cmdSetPowerState`	Switches radio nodes on or off (node individually or simultaneously for all nodes).
`cmdGetDutCurrent`	Provides the actual current consumption of a radio node (energy monitoring).
`cmdSetAttenuation`	Sets the insertion loss of the panel into the chain (steps: 1, 3, 11, 21, and 31 dB).
`cmdSetGpioConfig`	Configures the available *General Purpose Input Outputs (GPIOs)* per node slot.
`cmdSetGpio`	Switches the *GPIO* states between *LOW* or *HIGH*, e.g., for emulating external sensors.
`cmdSetSerialParameter`	Configures the serial interface parameters between the panel and the radio module.
`cmdSendTransparentData`	Sends arbitrary binary data to one or more radio nodes (emulation of radio data transmissions, *RIL* communication).

Table C.1: Selected emulation hardware commands and descriptions

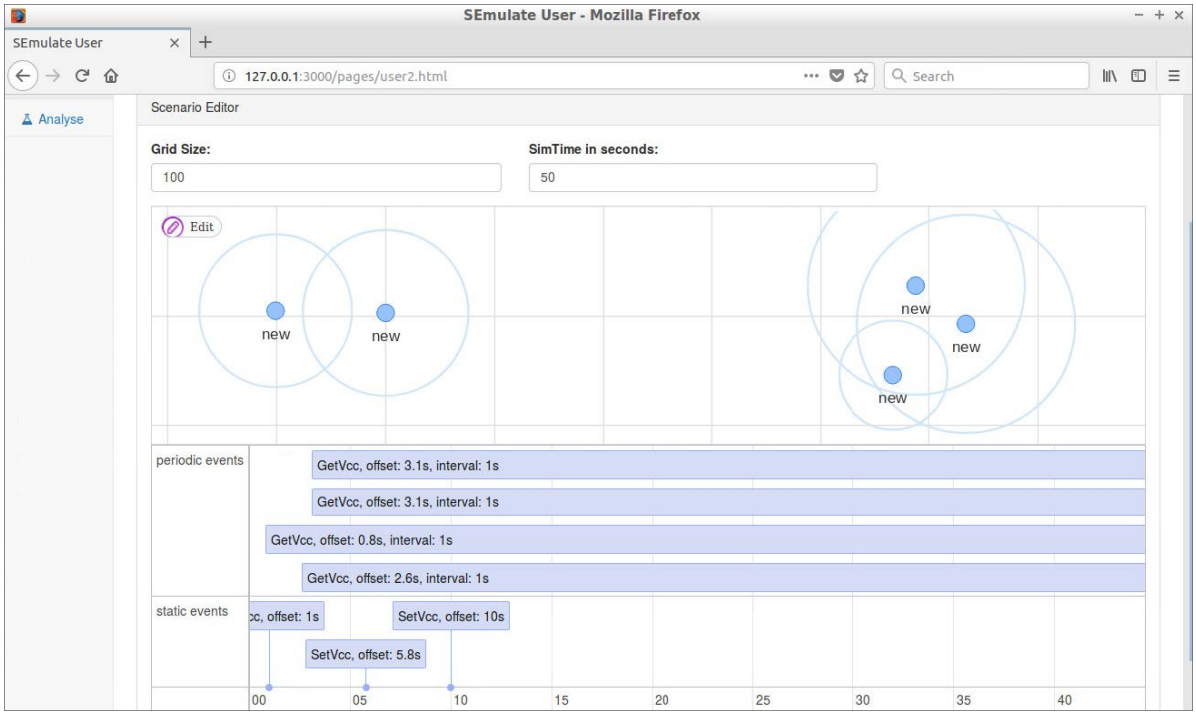


Figure C.2: Prototype of a Web-based radio emulation scenario configuration in *SEmulate*

D | Technologies, Protocols, and Standards

"The nice thing about standards is that you have so many to choose from. Furthermore, if you do not like any of them, you can just wait for next year's model."

Andrew S. Tanenbaum

BAUD

BAUD is the defacto standard unit of measurement of symbol or modulation rate to determine the communication speed over a data channel. It is defined as the number of symbol changes per second or pulses per second. If there are only two symbols (0 and 1), then BAUD and bit s^{-1} are equivalent.

Bluetooth LE

Bluetooth Low Energy (LE), *BLE* is a proprietary low-power radio technology for the *IoT*. It operates in the 2.4GHz *ISM* band based on *Frequency Shift Keying (FSK)* modulation using a *Time Division Multiple Access (TDMA) frequency hopping MAC*. The Bluetooth stack defines the *OSI layers* 1-2 and higher layer host components. The full specification including the link layer is defined by the Bluetooth *SIG*.

DECT ULE

DECT ULE is an international open standard for low-power wireless home automation. It operates around the 1.9GHz based on *FSK* modulation using *Frequency Division Multiple Access (FDMA)* and *TDMA MAC*. The protocol architecture defines the *OSI layers* 1-3. The link layer technology and protocol architecture is standardized in *ETSI TS* 102 939 [14].

EnOcean

EnOcean is a proprietary ultra-low-power wireless technology featuring a self-sufficient power supply with *energy harvesting* for wireless home and building automation. It operates in the license-free sub-GHz *ISM* band based on *ASK* and *FSK* modulation. The protocol architecture defines the *OSI layers* 1-3 and equipment profiles on the higher layers. The link layer technology and protocol architecture is standardized in *ISO/IEC* 14543-3-10 [8].

GNU Radio

GNU Radio is an open-source framework for development of *SDR* systems on general-purpose host systems. This streaming system provides a graphical development environment with a variety of *blocks* interconnected in *flow graphs* for complex signal processing, and interfaces to state-of-the-art wireless hardware.

IEEE 802.15.4

IEEE 802.15.4 is an open short range link layer protocol standard that has in particular shaped *WSNs* and the *IoT* from the very beginning. The protocol architecture defines the *OSI layers* 1-2. The spread of related technologies and communication protocols of this pre-dominant technology is exemplary given with the explanations for Figure 1.2. The full link layer specification for *LR-WPAN* is standardized by the *IEEE* (e.g., in [9]).

ISA100.11a

ISA100.11a designates a *WSN* technology and protocol stack for time-synchronized mesh architectures in industrial wireless automation. It operates in the 2.4GHz *ISM* band and defines *MAC* extensions using *IEEE 802.15.4*. On the higher layers, *6LoWPAN* and *UDP* are defined. The protocol architecture is standardized in *ANSI/ISA*-100.11a-2011 [10].

LoRaWAN

LoRaWAN is a system architecture and communication procedure for the *MAC* sublayer in *LPWANs*. *LoRaWAN* is used upon the *LoRa* (*long range* or *low radiation*) *PHY Chirp Spread Spectrum (CSS)* modulation from *Semtech Corporation* and thus defines the *OSI layers* 1-2. The technical characteristics are specified in *ETSI TR* 103 526 [15].

MATLAB/Simulink

MATLAB/Simulink is a proprietary block diagram environment for modeling and simulation dynamic real-world system behavior. It is often mentioned for wireless devices design, prototype development, and simulating signal processing applications, such as *SDR*.

MIOTY

MIOTY is a *LPWAN* radio technology standard for the *IoT*. It operates in the license-free sub-GHz *ISM* band based on *Ultra-Narrow Band (UNB) Minimum Shift Keying (MSK)* modulation and enables robust transmissions using *Telegram Splitting Multiple Access (TSMA)*. The MIOTY stack defined the *OSI layers* 1-2. The link layer technology and protocol architecture is standardized in *ETSI TS* 103 357 [16].

NB-IoT

Narrowband Internet of Things (NB-IoT) is a *LPWAN* radio technology standard for the *IoT*, developed by the *3rd Generation Partnership Project (3GPP)* for cellular devices and services. *NB-IoT* operates in multiple licensed frequency bands based on *QPSK* and *BPSK* modulation using *FDMA MAC*. The protocol architecture defines the *OSI layers* 1-3. The specification is part of the *ETSI/3GPP* Release 13 [17].

ns-2/ns-3

ns-3 [190] (network simulator version 3) is a C++ and Python based *DES* system for computer network simulation, primarily used in communication network research and teaching. It is the successor simulation framework of ns-2 (with the latest release version 2.35 in 2011), that is no longer in active development. ns-3 is often mentioned to be more accurate, because the software architecture is close to Linux with internal device driver and application interfaces. When looking at embedded, low-power wireless devices, this advantage has lost much of its practical relevance.

OMNeT

OMNeT++ [48] (Objective Modular Network Testbed in C++) is a generic simulation framework for research and development of complex distributed systems with a rapidly growing scientific community and a long period of active development. Many extensions, frameworks, and simulation models became available, most of them open-source, particularly for communication network areas, e.g., internet protocols, wireless networks, *WSNs*, mobile ad-hoc and mesh networks, or vehicular networks. On of the largest model framework is *INET*, which provides communication protocols, network devices, algorithmic models, and data structures.

PCAP

PCAP [158] is the defacto standard for network packet capturing (on the *NIC*), processing and visualization. It serves as an *API* for capturing network traffic. The operation system specific libraries provide also filtering engines for open-source and commercial network analysis tools, e.g., Wireshark [191].

RoSeNet

RoSeNet [115] is an abbreviation for a research project with the focus on development of methods and procedures for building *ro*bust and functionally reliable wireless *se*nsor-actuator *net*works. In this thesis context, RoSeNet refers to the hardware test platform and *NET* with hardware-based channel emulation capabilities for low-power wireless technologies. It features the analysis of large-scale *WSNs* with a size of up to 1000 radio modules.

SEmulate

SEmulate [54] is an abbreviation for a prototype implementation of the novel evaluation concept, introduced by this thesis, that seamlessly couples protocol *S*imulation and radio channel *E*mulation. Seamlessly coupled here means that the technology of hardware-based radio channel emulation is integrated into the network simulation without significantly changing the basic principle of running a *DES* scenario. The way the different systems are coupled follows exactly this basic principle.

Sigfox

Sigfox is a proprietary low-power radio technology for the *IoT* regarding *LPWANs*. It operates in the license-free sub-GHz *ISM* band based on *Ultra-Narrow Band (UNB) PSK* and *FSK* modulation using a random channel access *MAC*. The protocol architecture defines the *OSI layers* 1-4. The full specification comes from the SIGFOX S.A..

SmartRF

SmartRF is a proprietary sniffer software from *Texas Instruments* for wireless packet capture (on own radio chip hardware) and fundamental analysis (similar to *Wireshark*) for *Bluetooth LE*, *IEEE 802.15.4*, and *ZigBee*

Thread

Thread designates a *WSN* technology and protocol stack for the *IoT*. It is based on the *IEEE* 802.15.4 *PHY* and *MAC* and defines the *OSI layers* 1-4. Thread integrates the *IETF* protocols for *LLNs*, e.g., *6LoWPAN* and *IPv6*, among others. The protocol architecture is standardized by the Thread Group.

Wi-SUN

Wireless Smart Utility Network (Wi-SUN) is a wireless communication standard for connectivity of resource constrained devices in smart-grids. The communication technology is based on the *IEEE* 802.15.4g *PHY* and the IEEE 802.15.4e *MAC*. The protocol architecture defines the *OSI layers* 1-4. On the *NWK* custom profiles are specified, which focus on the *IETF* protocols for *LLNs*, e.g., *6LoWPAN* and *RPL*, among others. The protocol architecture is standardized by the *Wi-SUN* Alliance.

WIA-PA

Wireless Networks for Industrial Automation Process Automation (WIA-PA) is a protocol architecture for industrial wireless automation. It operates in the 2.4GHz *ISM* band and used the *IEEE 802.15.4 PHY* and *MAC* with standard *Beaconing*. The protocol architecture is standardized in *IEC* 62601 [11].

WirelessHART

WirelessHART designates a *WSN* technology and protocol stack for time-synchronized mesh architectures in industrial wireless automation. It operates in the 2.4GHz *ISM* band using a *TDMA MAC* with *IEEE 802.15.4*. The protocol architecture is standardized in *IEC* 62591 [12].

Wireshark

Wireshark [191] is the defacto standard *PCAP*-based packet sniffer and analyzer software. It provides analyzing packet headers and payload data for common protocols covering the whole internet protocol suite and countless proprietary protocols and technologies. It is easy to extend the analysis for any arbitrary protocol specification with custom *packet dissectors*.

Z-Wave

Z-Wave is a proprietary low-power and low-rate wireless technology for wireless home and building automation. It operates the license-free sub-GHz *ISM* band based on a *FSK* modulation transmitting short control messages within the network. The protocol architecture defines the *OSI layers* 1-2 and higher layers (*Transfer*, *Routing*, *Application*). The link layer technology is standardized in *ITU-T* G.9959 [13].

ZigBee

ZigBee designates a *WSN* technology and protocol stack for the *IoT* and wireless home automation. It is based on the *IEEE* 802.15.4 *PHY* and *MAC* and defines the whole communication stack. With the *IP* extensions (ZigBee *IP*) it integrates the *IETF* protocols for *LLNs*, e.g., *6LoWPAN* and *RPL*, among others. The protocol architecture is standardized by the ZigBee Alliance.

E | Acronyms

3GPP	3rd Generation Partnership Project.
6LoWPAN	IPv6 over Low-power s.
6TiSCH	IPv6 over the TSCH mode of IEEE 802.15.4e.
6top	over the mode of 802.15.4e Operation Sublayer.
AD	Analog-to-Digital.
ANSI	American National Standards Institute.
API	Application Programming Interface.
APL	Application Layer.
APS	Application Support Sub-Layer.
ARQ	Automatic Repeat Request.
ASK	Amplitude Shift Keying.
BER	Bit-Error Rate.
BI	Beacon Interval.
BLE	Bluetooth Low Energy.
BNC	Bayonet Neill–Concelman.
BO	Beacon Order.
BPSK	Binary Phase-Shift Keying.
CCA	Clear Channel Assessment.
CoAP	Constrained Application Protocol.
CPU	Central Processing Unit.
CR	Cognitive Radio.
CRC	Cyclic Redundancy Check.
CRSN	Cognitive Radio Sensor Network.
CSI	Channel State Information.
CSMA-CA	Carrier Sense Multiple Access Collision Avoidance.
CTI	Cross-Technology Interference.
CTS	Continuous Time Simulation.
CTW	Conservative Time Window.
DA	Digital-to-Analog.
DECT	Digital Enhanced Cordless Telecommunications.
DES	Discrete Event Simulation.
DLC	Data Link Control.
DLL	Data Link Layer.
DSME	Deterministic and Synchronous Multi-channel Extension.

DSP	Digital Signal Processor.
DSSS	Direct Sequence Spread Spectrum.
DUT	Device Under Test.
DVR	Distance Vector Routing.
ED	Energy Detection.
EPB	Enhanced Packet Block.
ESF	Event Stream Forwarder.
ETSI	European Telecommunications Standards Institute.
FDMA	Frequency Division Multiple Access.
FEC	Forward Error Correction.
FEL	Future Event List.
FES	Future Event Set.
FIFO	First In First Out.
FPGA	Field Programmable Gate Array.
FSK	Frequency Shift Keying.
GPIO	General Purpose Input Output.
GRC	GNU Radio Companion.
GTS	Guaranteed Time Slot.
GUI	Graphical User Interface.
GVT	Global Virtual Time.
HAL	Hardware Abstraction Layer.
HIL	Hardware-in-the-Loop.
ICI	Inter-Channel Interference.
IDB	Interface Description Block.
IEC	International Electrotechnical Commission.
IEEE	Institute of Electrical and Electronics Engineers.
IETF	Internet Engineering Task Force.
IFS	Inter Frame Spacing.
IO	Input/Output.
IoT	Internet of Things.
IP	Internet Protocol.
IPC	Inter Process Communication.
IPv6	Internet Protocol version 6.
ISA	International Society of Automation.
ISI	Inter Symbol Interference.
ISM	Industrial, Scientific and Medical.
ISO	International Organization for Standardization.
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector.
JTAG	Joint Test Action Group.
L2CAP	Logical Link Control and Adaptation Protocol.
LAN	Local Area Network.

LIFS	Long Inter Frame Spacing.
LLC	Logical Link Control.
LLN	Low-power and Lossy Network.
LoRaWAN	Long Range Wide Area Network.
LoS	Line-of-Sight.
LP	Linear Program.
LPC	Local Procedure Call.
LPWAN	Low Power Wide Area Network.
LQI	Link Quality Indicator.
LR-WPAN	Low Rate Wireless Personal Area Network.
MAC	Medium Access Control.
MILP	Mixed Integer Linear Programming.
MIMO	Multiple Input Multiple Output.
MINLP	Mixed Integer Non-Linear Programming.
ML	Machine Learning.
MPDU	Medium Access Control Protocol Data Unit.
MSK	Minimum Shift Keying.
NB-IoT	Narrowband Internet of Things.
NET	Network Emulation Testbed.
NIC	Network Interface Controller.
NSC	Network Simulation Cradle.
NTP	Network Time Protocol.
NWK	Network Control Layer.
OFDM	Orthogonal Frequency Division Multiplex.
OSI	Open Systems Interconnection.
PaaS	Platform as a Service.
PAN	Personal Area Network.
PCAP	Packet Capture.
PD	Physical Layer Data Service.
PDES	Parallel Discrete Event Simulation.
PDU	Protocol Data Unit.
PER	Packet Error Rate.
PHY	Physical Layer.
PIB	Personal Area Network Information Base.
PLME	Physical Layer Management Entity.
PPDU	Physical Layer Protocol Data Unit.
PSDU	Physical Layer Service Data Unit.
PSK	Phase-Shift Keying.
PTP	Precision Time Protocol.
QoS	Quality of Service.
QPSK	Quadrature Phase-Shift Keying.
REST	Representational State Transfer.

RF	Radio Frequency.
RFC	Request for Comments.
RIL	Radio-in-the-Loop.
RPC	Remote Procedure Call.
RPL	Routing Protocol for Low-power and Lossy Networks.
RSSI	Received Signal Strength Indicator.
RSV	Real-time-Shift Vector.
RTT	Round Trip Time.
RX	Receiving.
SAP	Service Access Point.
SDR	Software-Defined Radio.
SDU	Service Data Unit.
SHB	Section Header Block.
SIFS	Short Inter Frame Spacing.
SIG	Special Interest Group.
SNR	Signal-to-Noise-Ratio.
SoC	System-on-Chip.
TCP	Transmission Control Protocol.
TDMA	Time Division Multiple Access.
TR	Technical Report.
TS	Technical Specification.
TSCH	Time Slotted Channel Hopping.
TSMA	Telegram Splitting Multiple Access.
TX	Transmitting.
UART	Universal Asynchronous Receiver and Transmitter.
UDG	Unit Disk Graph.
UDP	User Datagram Protocol.
UDS	Unix Domain Socket.
ULE	Ultra Low Energy.
VANET	Vehicular Ad Hoc Network.
VM	Virtual Machine.
VTB	Virtual Testbed.
WAN	Wide Area Network.
Wi-SUN	Wireless Smart Utility Network.
WLAN	Wireless Local Area Network.
WPAN	Wireless Personal Area Network.
WSN	Wireless Sensor Network.

Bibliography

[1] R. Beuran: *Introduction to Network Emulation*. 1st ed. Pan Stanford Publishing, Nov. 2012. 426 pp. ISBN: 978-9814310918.

[2] A. Nikoukar, S. Raza, A. Poole, M. Gunes, and B. Dezfouli: "Low-Power Wireless for the Internet of Things: Standards and Applications". In: *IEEE Access* 6 (2018), pp. 67893–67926. DOI: 10.1109/access.2018.2879189.

[3] S. Kharb and A. Singhrova: "Review of Industrial Standards for Wireless Sensor Networks". In: *Next-Generation Networks*. Ed. by D. K. Lobiyal, V. Mansotra, and U. Singh. Singapore: Springer Singapore, 2018, pp. 77–87. ISBN: 978-981-10-6005-2.

[4] H. Sharma and S. Sharma: "A Review of Sensor Networks: Technologies and Applications". In: *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*. 2014, pp. 1–4. DOI: 10.1109/RAECS.2014.6799579.

[5] A. Ikpehai et al.: "Low-Power Wide Area Network Technologies for Internet-of-Things: A Comparative Review". In: *IEEE Internet of Things Journal* 6.2 (2019), pp. 2225–2240. DOI: 10.1109/JIOT.2018.2883728.

[6] M. Ahmad, A. Ishtiaq, M. A. Habib, and S. H. Ahmed: "A Review of Internet of Things (IoT) Connectivity Techniques". In: *Recent Trends and Advances in Wireless and IoT-enabled Networks*. Springer International Publishing, 2019, pp. 25–36. DOI: 10.1007/978-3-319-99966-1_3.

[7] N. Poursafar, M. E. E. Alahi, and S. Mukhopadhyay: "Long-Range Wireless Technologies for IoT Applications: A Review". In: *2017 Eleventh International Conference on Sensing Technology (ICST)*. IEEE, Dec. 2017. DOI: 10.1109/icsenst.2017.8304507.

[8] ISO Central Secretary: *Information technology — Home electronic systems (HES) architecture — Part 3-10: Wireless short-packet (WSP) protocol optimized for energy harvesting — Architecture and lower layer protocols*. en. Standard ISO/IEC 14543-3-10. Geneva, CH: International Organization for Standardization, 2020. URL: https://www.iso.org/standard/80934.html.

[9] 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™-2006. The Institute of Electrical and Electronics Engineers, Inc., Sept. 2006. 320 pp. DOI: 10.1109/IEEESTD.2006.232110.

[10] ISA: *ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications*. Standard. International Society of Automation, 2011. URL: https://www.isa.org/products/ansi-isa-100-11a-2011-wireless-systems-for-industr.

[11] IEC: *Industrial networks - Wireless communication network and communication profiles - WIA-PA*. en. International Standard IEC 62601:2015. Geneva, CH: International Electrotechnical Commission, 2015. URL: https://www.vde-verlag.de/iec-normen/222404/iec-62601-2015.html.

[12] IEC: *Industrial networks - Wireless communication network and communication profiles - WirelessHART*. en. International Standard IEC 62591:2016. Geneva, CH: International Electrotechnical Commission, 2016. URL: https://www.vde-verlag.de/iec-normen/222660/iec-62591-2016.html.

[13] ITU-T: *Short range narrow-band digital radiocommunication transceivers – PHY, MAC, SAR and LLC layer specifications*. Recommendation ITU-T G.9959. International Telecommunication Union, 2015. URL: https://www.itu.int/rec/T-REC-G.9959-201501-I.

[14] ETSI: *Digital Enhanced Cordless Telecommunications (DECT); Ultra Low Energy (ULE); Machine to Machine Communications; Part 1: Home Automation Network (phase 1)*. en. Technical Specification (TS) ETSI TS 102 939-1. V1.1.1. European Telecommunications Standards Institute, 2015. URL: https://www.etsi.org/deliver/etsi_ts/102900_102999/10293902/01.01.01_60/ts_10293902v010101p.pdf.

[15] ETSI: *System Reference document (SRdoc);Technical characteristics for Low Power Wide Area NetworksChirp Spread Spectrum (LPWAN-CSS) operating in the UHF spectrum below 1 GHz*. en. Technical Report (TR) ETSI TS 103 526. V1.1.1. European Telecommunications Standards Institute, 2018. URL: https://www.etsi.org/deliver/etsi_tr/103500_103599/103526/01.01.01_60/tr_103526v010101p.pdf.

[16] ETSI: *Short Range Devices; Low Throughput Networks (LTN); Protocols for radio interface A*. en. Technical Specification (TS) ETSI TS 103 357. V1.1.1. European Telecommunications Standards Institute, 2018. URL: https://www.etsi.org/deliver/etsi_ts/103300_103399/103357/01.01.01_60/ts_103357v010101p.pdf.

[17] ETSI: *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification*. Technical Specification (TS) 36.141. Version 13.6.0 Release 13. European Telecommunications Standards Institute, 2017. URL: https://www.etsi.org/deliver/etsi_ts/136100_136199/136141/13.06.00_60/ts_136141v130600p.pdf.

[18] X. Zhang: "Reconfigurable Medium Access Control Protocols for Wireless Networks". PhD thesis. Aachen, Germany: RWTH Aachen University, 2014.

[19] P. De Mil et al.: “snapMac: A generic MAC/PHY architecture enabling flexible MAC design”. In: *Ad Hoc Networks* 17 (June 2014), pp. 37–59. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2014.01.004.

[20] A. B. Ozgur, O. Karli, and O. Ergul: “Cognitive Radio Sensor Networks”. In: *Network, IEEE* 23.4 (2009), pp. 34–40. DOI: 10.1109/mnet.2009.5191144.

[21] T. Esemann and H. Hellbrück: “Integrated Low-Power SDR enabling Cognitive IEEE 802.15.4 Sensor Nodes”. In: *Proceedings of the 8th Karlsruhe Workshop on Software Radios*. 2014.

[22] G. Joshi, S. Nam, and S. Kim: “Cognitive Radio Wireless Sensor Networks: Applications, Challenges and Research Trends”. In: *Sensors* 13.9 (Aug. 2013), pp. 11196–11228. DOI: 10.3390/s130911196.

[23] A. G. Ramonet and T. Noguchi: “IEEE 802.15.4 Now and Then: Evolution of the LR-WPAN Standard”. In: *Proceedings of the 22nd International Conference on Advanced Communication Technology (ICACT)*. IEEE, Feb. 2020. DOI: 10.23919/icact48636.2020.9061514.

[24] IEEE Standards Association: *IEEE Standard for Local and Metropolitan Area Networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC Sublayer*. IEEE Std 802.15.4e-2012 - Amendment to IEEE Std 802.15.4™-2011. The Institute of Electrical and Electronics Engineers, Inc., Apr. 2012. 225 pp. DOI: 10.1109/IEEESTD.2012.6185525.

[25] IEEE Standards Association: *IEEE Standard for Local and Metropolitan Area Networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4-2011 - Revision of IEEE Std. 802.15.4™-2006. The Institute of Electrical and Electronics Engineers, Inc., Sept. 2011. 314 pp. DOI: 10.1109/IEEESTD.2011.6012487.

[26] IEEE Standards Association: *IEEE Standard for Local and Metropolitan Area Networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 3: Physical Layer (PHY) Specifications for Low-Data-Rate, Wireless, Smart Metering Utility Networks*. IEEE Std 802.15.4g-2012 - Amendment to IEEE Std 802.15.4™-2011. The Institute of Electrical and Electronics Engineers, Inc., Apr. 2012. 252 pp. DOI: 10.1109/IEEESTD.2012.6190698.

[27] S. Petersen and S. Carlsen: “WirelessHART Versus ISA100.11a: The Format War Hits the Factory Floor”. In: *IEEE Industrial Electronics Magazine* 5.4 (Dec. 2011), pp. 23–34. DOI: 10.1109/mie.2011.943023.

[28] W. Liang et al.: “Survey and Experiments of WIA-PA Specification of Industrial Wireless Network”. In: *Wireless Communications and Mobile Computing* 11.8 (May 2010), pp. 1197–1212. DOI: 10.1002/wcm.976.

[29] R. K. Ghosh: “Low Power Communication Protocols: ZigBee, 6LoWPAN and ZigBee IP”. In: *Wireless Networking and Mobile Data Management*. Springer Singapore, 2017, pp. 147–177. DOI: 10.1007/978-981-10-3941-6_6.

[30] G. Montenegro, J. Huii, D. Culler, and N. Kushalnagar: *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. 4944. Sept. 2007. 30 pp. DOI: 10.17487/RFC4944.

[31] Q. Wang, X. Vilajosana, and T. Watteyne: *6TiSCH Operation Sublayer (6top) Protocol (6P)*. 8480. Nov. 2018. 50 pp. DOI: 10.17487/RFC8480.

[32] Z. Shelby, K. Hartke, and C. Bormann: *The Constrained Application Protocol (CoAP)*. 7252. June 2014. 112 pp. DOI: 10.17487/RFC7252.

[33] X. Vilajosana, K. Pister, and T. Watteyne: *Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration*. 8180. May 2017. 28 pp. DOI: 10.17487/RFC8180.

[34] J. Nieminen et al.: *IPv6 over BLUETOOTH(R) Low Energy*. 7668. Oct. 2015. 21 pp. DOI: 10.17487/RFC7668.

[35] P. B. Mariager, J. T. Petersen, Z. Shelby, M. van de Logt, and D. Barthel: *Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)*. 8105. May 2017. 22 pp. DOI: 10.17487/RFC8105.

[36] M. Imran, A. M. Said, and H. Hasbullah: "A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks". In: *Proceedings of the 4th International Symposium in Information Technology (ITSim)*. IEEE, 2010. DOI: 10.1109/ITSIM.2010.5561571.

[37] G. Z. Papadopoulos, K. Kritsis, A. Gallais, P. Chatzimisios, and T. Noel: "Performance Evaluation Methods in Ad Hoc and Wireless Sensor Networks: A Literature Study". In: *IEEE Communications Magazine* 54.1 (2016), pp. 122–128. DOI: 10.1109/mcom.2016.7378437.

[38] K. Wehrle, M. Günes, and J. Gross: *Modeling and Tools for Network Simulation*. Springer Science & Business Media, 2010. DOI: 10.1007/978-3-642-12331-3.

[39] J. Burbank, W. Kasch, and J. Ward: *An Introduction to Network Modeling and Simulation for the Practical Engineer*. 1st ed. ComSoc Guides to Communications Technologies. John Wiley & Sons, Oct. 2011. 216 pp. ISBN: 978-0-470-46726-8. DOI: 10.1002/9781118063651.

[40] G. Coulson et al.: "Flexible Experimentation in Wireless Sensor Networks". In: *Communications of the ACM* 55.1 (Jan. 2012), pp. 82–90. ISSN: 0001-0782. DOI: 10.1145/2063176.2063198.

[41] A. K. Dwivedi and O. P. Vyas: "An Exploratory Study of Experimental Tools for Wireless Sensor Networks". In: *Wireless Sensor Network* 3.7 (2011), pp. 215–240. DOI: 10.4236/wsn.2011.37025.

[42] K. Kritsis, G. Z. Papadopoulos, A. Gallais, P. Chatzimisios, and F. Theoleyre: "A Tutorial on Performance Evaluation and Validation Methodology for Low-Power and Lossy Networks". In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 1799–1825. DOI: 10.1109/comst.2018.2820810.

[43] M. N. Jambli, H. Lenando, K. Zen, S. M. Suhaili, and A. Tully: "Simulation Tools for Mobile Ad-hoc Sensor Networks: A State-of-the-Art Survey". In: *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)* (Nov. 2012). DOI: 10.1109/acsat.2012.84.

[44] G. Chelius and J.-M. Gorce: "Impact of the Physical Layer Modeling on the Accuracy and Scalability of Wireless Network Simulation". In: *SIMULATION: Transactions of The Society for Modeling and Simulation International* 85 (Aug. 2009). DOI: 10.1177/0037549709106633.

[45] S. T.R. and M. P. Sebastian: "A Classification of the Debugging Techniques of Wireless Sensor Networks". In: *Proceedings of the International Conference on Advances in Computing and Communications*. IEEE, Aug. 2012. ISBN: 978-0-7695-4723-7. DOI: 10.1109/icacc.2012.12.

[46] W. Du, F. Mieyeville, D. Navarro, I. O'Connor, and L. Carrel: "Modeling and Simulation of Networked Low-Power Embedded Systems: A Taxonomy". In: *EURASIP Journal on Wireless Communications and Networking* 2014.1 (2014), pp. 1–12. DOI: 10.1186/1687-1499-2014-106.

[47] S. Lohier, A. Rachedi, E. Livolant, and I. Salhi: "Wireless Sensor Network simulators relevance compared to a real IEEE 802.15.4 Testbed". In: *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE, July 2011. DOI: 10.1109/iwcmc.2011.5982734.

[48] A. Virdis and M. Kirsche, eds.: *Recent Advances in Network Simulation*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-12842-5.

[49] V. Srivastava and M. Motani: "Cross-Layer Design and Optimization in Wireless Networks". In: *Cognitive Networks: Towards Self-Aware Networks*. Ed. by Q. Mahmoud. John Wiley & Sons, Ltd., July 2007. Chap. 6, pp. 121–146. ISBN: 978-0-470-06196-1. DOI: 10.1002/9780470515143.ch6.

[50] S. N. Khan, M. A. Kalil, and A. Mitschele-Thiel: "crSimulator: A Discrete Simulation Model for Cognitive Radio Ad Hoc Networks in OMNeT++". In: *Proceedings of the 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE. 2013, pp. 1–7. DOI: 10.1109/WMNC.2013.6549029.

[51] R. Massin, C. Lamy-Bergot, C. J. L. Martret, and R. Fracchia: "OMNeT++-Based Cross-Layer Simulator for Content Transmission over Wireless Ad Hoc Networks". In: *EURASIP Journal on Wireless Communications and Networking* 2010.1 (Jan. 2010). DOI: 10.1155/2010/502549.

[52] J. Mittag, S. Papanastasiou, H. Hartenstein, and E. G. Ström: "Enabling Accurate Cross-Layer PHY/MAC/NET Simulation Studies of Vehicular Communication Networks". In: *Proceedings of the IEEE* 99.7 (2011), pp. 1311–1326. DOI: 10.1109/jproc.2010.2103291.

[53] A.-S. Tonneau, N. Mitton, and J. Vandaele: "A Survey on (mobile) Wireless Sensor Network Experimentation Testbeds". In: *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, May 2014. DOI: 10.1109/dcoss.2014.41.

[54] S. Böhm and H. König: "SEmulate: Seamless Network Protocol Simulation and Radio Channel Emulation for Wireless Sensor Networks". In: *Proceedings of the 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, 2019, p. 8. ISBN: 978-3-903176-13-3. DOI: 10.23919/WONS.2019.8795495.

[55] S. Böhm and M. Kirsche: "Unifying Radio-in-the-Loop Channel Emulation and Network Protocol Simulation to Improve Wireless Sensor Network Evaluation". In: *Simulation Science*. Ed. by M. Baum et al. Springer International Publishing, 2018, pp. 219–238. ISBN: 978-3-319-96271-9. DOI: 10.1007/978-3-319-96271-9_14.

[56] S. Böhm and H. König: "Split-Protocol-Stack Wireless Network Emulation: Enabling PHY Modeling Diversity with Software-Radio-in-the-Loop". In: *Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '21)*. ACM Press, 2021, p. 8. ISBN: 978-1-4503-9080-4/21/11. DOI: 10.1145/3479242.3487319.

[57] S. Böhm and H. König: "Real-Time-Shift: Pseudo-Real-Time Event Scheduling for the Split-Protocol-Stack Radio-in-the-Loop Emulation". In: *Proceedings of the 25th International Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. ACM, Oct. 2022. DOI: 10.1145/3551659.3559057.

[58] S. Böhm and M. Kirsche: "Looking into Hardware-in-the-Loop Coupling of OMNeT++ and RoSeNet". In: *Proceedings of the 2nd International OMNeT++ Community Summit (OMNeT 2015)*. Zurich, Switzerland, 2016. URL: http://arxiv.org/abs/1509.03558.

[59] D. Kotz et al.: "Experimental Evaluation of Wireless Simulation Assumptions". In: *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. ACM Press, 2004. DOI: 10.1145/1023663.1023679.

[60] J. Banks, J. S. C. II, B. L. Nelson, and D. M. Nicol: *Discrete-Event System Simulation, 5th Edition*. 5th ed. Pearson Education, 2010. ISBN: 978-1-292-02437-0.

[61] A. M. Law: *Simulation Modeling & Analysis*. 5th ed. New York, NY, USA: McGraw-Hill, 2015.

[62] F. J. Suárez, P. Nuño, J. C. Granda, and D. F. García: "Computer Networks Performance Modeling and Simulation". In: *Modeling and Simulation of Computer Networks and Systems*. Ed. by M. S. Obaidat, P. Nicopolitidis, and F. Zarai. Boston: Morgan Kaufmann, 2015. Chap. 7, pp. 187–223. ISBN: 978-0-12-800887-4. DOI: 10.1016/B978-0-12-800887-4.00007-9.

[63] K. Kuladinithi et al.: “Teaching Modelling and Analysis of Communication Networks using OMNeT++ Simulator”. In: *Proceedings of the 5th International OMNeT++ Community Summit*. Ed. by A. F\”orster, A. Udugama, A. Virdis, and G. Nardini. Vol. 56. EPiC Series in Computing. EasyChair, 2018, pp. 111–123. DOI: 10.29007/xtt6.

[64] R. M. Fujimoto, K. S. Perumalla, and G. F. Riley: *Network Simulation*. 1st ed. Morgan & Claypool Publishers, 2007. 66 pp. ISBN: 978-1598291100.

[65] D. Mahrenholz and S. Ivanov: “Real-time Network Emulation with ns-2”. In: *Proceedings of the 8th International Symposium on Distributed Simulation and Real-Time Applications*. IEEE. 2004, pp. 29–36. DOI: 10.1109/DS-RT.2004.34.

[66] M. Tüxen, I. Rüngeler, and E. P. Rathgeb: “Interface Connecting the INET Simulation Framework with the Real World”. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques (SIMUTools)*. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, 40:1–40:6. DOI: 10.1145/1416222.1416267.

[67] J. Bélanger, P. Venne, and J.-N. Paquin: *The What, Where, and Why of Real-Time Simulation*. Jan. 2010.

[68] A. Goldsmith: *Wireless communications*. Cambridge university press, 2005.

[69] J. Pavon and S. Chio: “Link Adaptation Strategy for IEEE 802.11 WLAN via Received Signal Strength Measurement”. In: *IEEE International Conference on Communications ICC*. IEEE, Feb. 2003. DOI: 10.1109/icc.2003.1204534.

[70] E. Municio et al.: “Simulating 6TiSCH Networks”. In: *Transactions on Emerging Telecommunications Technologies* 30 (Sept. 2018). DOI: 10.1002/ett.3494.

[71] S.-H. Yang: *Wireless Sensor Networks Principles, Design and Applications*. Signals and Communication Technology. Springer London, 2014. DOI: 10.1007/978-1-4471-5505-8.

[72] M. Kirsche and J. Hartwig: “A 6LoWPAN Model for OMNeT++: Poster Abstract”. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. SimuTools ’13. Cannes, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 330–333. ISBN: 9781450324649. DOI: 10.4108/icst.simutools.2013.251710.

[73] A. Rayes and S. Salam: “IoT Protocol Stack: A Layered View”. In: *Internet of Things From Hype to Reality*. Springer International Publishing, Oct. 2016, pp. 93–138. DOI: 10.1007/978-3-319-44860-2_5.

[74] H. M. A. Fahmy: *Wireless Sensor Networks: Concepts, Applications, Experimentation and Analysis*. Springer, 2016.

[75] S. C. Mukhopadhyay and N. Suryadevara: *Internet of Things: Challenges and Opportunities*. Springer, 2014. DOI: 10.1007/978-3-319-04223-7.

[76] K. P. Naik and U. R. Joshi: "Performance Analysis of Constrained Application Protocol using Cooja Simulator in Contiki OS". In: *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*. IEEE, July 2017. DOI: 10.1109/icicict1.2017.8342622.

[77] A. Yaqoob, M. A. Ashraf, F. Ferooz, A. H. Butt, and Y. D. Khan: "WSN Operating Systems for Internet of Things(IoT): A Survey". In: *2019 International Conference on Innovative Computing (ICIC)*. IEEE, Nov. 2019. DOI: 10.1109/icic48496.2019.8966731.

[78] P. Levis, N. Lee, M. Welsh, and D. Culler: "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications". In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM. New York, NY, USA: ACM Press, 2003, pp. 126–137. DOI: 10.1145/958491.958506.

[79] P. Levis et al.: "TinyOS: An Operating System for Sensor Networks". In: *Ambient Intelligence*. Ed. by W. Weber, J. M. Rabaey, and E. Aarts. Springer Berlin Heidelberg, pp. 115–148. DOI: 10.1007/3-540-27139-2_7.

[80] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt: "Cross-Level Sensor Network Simulation with COOJA." In: *Proceedings of the 31th Annual IEEE International Conference on Local Computer Networks (LCN)*. Tampa, FL, USA: IEEE Computer Society, 2006, pp. 641–648. DOI: 10.1109/LCN.2006.322172.

[81] A. Dunkels, B. Gronvall, and T. Voigt: "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors". In: *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462. DOI: 10.1109/LCN.2004.38.

[82] R. de Paz Alberola and D. Pesch: "AvroraZ: Extending Avrora with an IEEE 802.15.4 Compliant Radio Chip Model". In: *Proceedings of the 3nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*. PM2HW2N '08. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2008, pp. 43–50. ISBN: 9781605582399. DOI: 10.1145/1454630.1454637.

[83] M. F. Oleg Hahm: *The DES Testbed virtualization framework*. Sept. 1, 2021. URL: https://github.com/des-testbed/desvirt.

[84] E. Baccelli et al.: "RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT". In: *IEEE Internet of Things Journal* 5.6 (Dec. 2018), pp. 4428–4440. DOI: 10.1109/jiot.2018.2815038.

[85] H. Sundani, H. Li, V. K. Devabhaktuni, M. Alam, and P. Bhattacharya: "Wireless Sensor Network Simulators A Survey and Comparisons". In: *International Journal Of Computer Networks (IJCN)* 2 (2011).

[86] E. Egea-Lopez, J. Vales-Alonso, A. S. Martinez-Sala, P. Pavon-Marino, and J. García-Haro: "Simulation Tools for Wireless Sensor Networks". In: *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. Society for Modeling and Simulation International, SCS, 2005, p. 24.

[87] E. Weingartner, H. vom Lehn, and K. Wehrle: "A Performance Comparison of Recent Network Simulators". In: *Proceedings of the 2009 IEEE International Conference on Communications*. IEEE, June 2009. DOI: 10.1109/icc.2009.5198657.

[88] T. A. R. V, E. Gamess, and D. Thornton: "A Survey of Wireless Network Simulation and/or Emulation Software for use in Higher Education". In: *Proceedings of the 2021 ACM Southeast Conference*. ACM, Apr. 2021. DOI: 10.1145/3409334.3452066.

[89] G. Chengetanai and G. B. O'Reilly: "Survey on Simulation Tools for Wireless Mobile Ad Hoc Networks". In: *Proceedings of the 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, Mar. 2015. DOI: 10.1109/icecct.2015.7226167.

[90] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg: "Mininet-WiFi: Emulating Software-Defined Wireless Networks". In: *Proceedings of the 11th International Conference on Network and Service Management (CNSM)*. IEEE, Nov. 2015. DOI: 10.1109/cnsm.2015.7367387.

[91] OpenSim Ltd.: *OMNeT++ Discrete Event Simulator*. 2021. URL: https://www.omnetpp.org/ (visited on 05/21/2021).

[92] nsnam: *ns-3 Network Simulator*. 2021. URL: https://www.nsnam.org/ (visited on 05/21/2021).

[93] S. Unterschütz, A. Weigel, and V. Turau: "Cross-Platform Protocol Development Based on OMNeT++". In: *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques (SIMUTools)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2012, pp. 278–282. DOI: 10.4108/icst.simutools.2012.247711.

[94] M. Slabicki, G. Premsankar, and M. D. Francesco: "Adaptive Configuration of Lora Networks for Dense IoT Deployments". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Apr. 2018. DOI: 10.1109/noms.2018.8406255.

[95] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso: "Do LoRa Low-Power Wide-Area Networks Scale?" In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. ACM, Nov. 2016. DOI: 10.1145/2988287.2989163.

[96] M. Kirsche and M. Schnurbusch: "A New IEEE 802.15.4 Simulation Model for OMNeT++ / INET". In: *Proceedings of the 1st International OMNeT++ Community Summit (OMNeT)*. Sept. 2014. URL: http://arxiv.org/abs/1409.1177.

[97] M. Kirsche: "Selected System Models - IEEE 802.15.4". In: *Modeling and Tools for Network Simulation*. Ed. by K. Wehrle, M. Guenes, and J. Gross. Springer, Mar. 2010. Chap. 12.3, pp. 276–303. DOI: 10.1007/978-3-642-12331-3.

[98] S. Khan et al.: "Reliability of Network Simulators and Simulation Based Research". In: *Proceedings of the 24th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, Sept. 2013. DOI: 10.1109/pimrc.2013.6666127.

[99] Y. M. Amin and A. T. Abdel-Hamid: "A Simulation Model of IEEE 802.15.4 GTS Mechanism and GTS Attacks in OMNeT++ / MiXiM + NETA". In: *Computer and Information Science* 11.1 (Jan. 2018), p. 78. DOI: 10.5539/cis.v11n1p78.

[100] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mario, and J. Garcia-Haro: "Simulation Scalability Issues in Wireless Sensor Networks". In: *IEEE Communications Magazine* 44.7 (Sept. 2006), pp. 64–73. DOI: 10.1109/mcom.2006.1668384.

[101] R. Fujimoto: "Parallel and Distributed Simulation". In: *Proceedings of the 2015 Winter Simulation Conference (WSC)*. 2015, pp. 45–59. DOI: 10.1109/WSC.2015.7408152.

[102] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens: "An Experimental Study of Cross-Technology Interference in In-Vehicle Wireless Sensor Networks". In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, Nov. 2016. DOI: 10.1145/2988287.2989141.

[103] A. Iyer, C. Rosenberg, and A. Karnik: "What is the right model for wireless channel interference?" In: *Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks - QShine '06*. ACM Press, 2006. DOI: 10.1145/1185373.1185376.

[104] U. Noreen, A. Bounceur, and L. Clavier: "Modeling Interference for Wireless Sensor Network Simulators". In: *Proceedings of the International Conference on Future Networks and Distributed Systems*. ACM, July 2017. DOI: 10.1145/3102304.3102347.

[105] M. Kropff, T. Krop, M. Hollick, P. S. Mogre, and R. Steinmetz: "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)*. IEEE. 2006, 6–pp. DOI: 10.1109/tridnt.2006.1649182.

[106] H. Hellbrück et al.: “Using and Operating Wireless Sensor Network Testbeds with WISEBED”. In: *Proceedings of the 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop*. IEEE. IEEE Press, June 2011, pp. 171–178. DOI: 10.1109/Med-Hoc-Net.2011.5970485.

[107] G. Judd: “Using Physical Layer Emulation to Understand and Improve Wireless Networks”. PhD thesis. Intel, 2006.

[108] P. Karimi, S. Mukherjee, J. Kolodziejski, I. Seskar, and D. Raychaudhuri: “Measurement Based Mobility Emulation Platform for Next Generation Wireless Networks”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2018, pp. 330–335. DOI: 10.1109/INFCOMW.2018.8407025.

[109] S. Koslowski, M. Braun, J. Elsner, and F. K. Jondral: “Wireless Networks In-the-Loop: Emulating an RF front-end in GNU Radio”. In: *SDR Forum 2010 European Reconfigurable Radio Technologies Workshop, Mainz, June 25, 2010*. 2010.

[110] E. Grayver: *Implementing Software Defined Radio*. Ed. by E. Grayver. Springer, 2013. DOI: 10.1007/978-1-4419-9332-8.

[111] J. W. Jung and M. A. Ingram: “An RF Channel Emulator-Based Testbed for Cooperative Transmission Using Wireless Sensor Devices”. In: *Proceedings of the Second International Conference on New Technologies, Mobility and Security (NTMS)*. 2008. DOI: 10.1109/NTMS.2008.ECP.22.

[112] J. D. Beshay et al.: “Wireless Networking Testbed and Emulator (WiNeTestEr)”. In: *Computer Communications* 73 (2016), pp. 99–107. DOI: 10.1145/2641798.2641809.

[113] R. Beuran, J. Nakata, Y. Tan, and Y. Shinoda: “Emulation Testbed for IEEE 802.15.4 Networked Systems”. In: *IEICE Transactions on Communications* E95-B.9 (Sept. 2012), pp. 2892–2905.

[114] J. Elsner, M. Braun, S. Nagel, K. Nagaraj, and F. K. Jondral: “Wireless Networks in-the-Loop: Software Radio as the Enabler”. In: *Proceedings of the Software Defined Radio Forum Technical Conference*. 2009.

[115] M. Ludwig et al.: *Verbundprojekt: Entwicklung von Methoden und Verfahren für den Aufbau von robusten und funktionssicheren drahtlosen Sensor-Aktor-Netzwerken*. Tech. rep. dresden elektronik Ingenieurtechnik GmbH, Fraunhofer-Institut für Integrierte Schaltungen, Dresden, Germany, June 2012.

[116] K. C. Borries, G. Judd, D. D. Stancil, and P. Steenkiste: “FPGA-Based Channel Simulator for a Wireless Network Emulator”. In: *Proceedings of the 69th Vehicular Technology Conference (VTC)*. IEEE, 2009. DOI: 10.1109/VETECS.2009.5073565.

[117] H. Wu, S. Member, Q. Luo, P. Zheng, and L. M. Ni: “VMNet: Realistic Emulation of Wireless Sensor Networks”. In: *Parallel and Distributed Systems, IEEE Transactions on* 18.2 (2007), pp. 277–288. DOI: 10.1109/tpds.2007.33.

[118] J. Flynn, H. Tewari, and D. O'Mahony: "JEmu: A Real-Time Emulation System for Mobile Ad-Hoc Networks". In: *Proceedings of the First Joint IEI/IEE Symposium on Telecommunications Systems Research*. Network Telecommunications Research Group (NTRG). IEEE, 2001.

[119] H. Onishi, F. Mlinarsky, F. Watanabe, and C. Velasquez: "Wireless Technology Assessment with Radio Channel Emulator". In: *Proceedings of the 20th ITS World Congress*. ITS America. Nashville, Tennessee, USA, Apr. 2013.

[120] J. Matai, P. Meng, L. Wu, B. Weals, and R. Kastner: "Designing a Hardware in the Loop Wireless Digital Channel Emulator for Software Defined Radio". In: *Proceedings of the International Conference on Field-Programmable Technology (FPT)*. IEEE. 2012, pp. 206–214. DOI: 10.1109/fpt.2012.6412135.

[121] D. Comer, R. H. Karandikar, A. Rastegarnia, F. Rouzbeh, and P. C. Sruthi: "WIST: Wi-SUN FAN Protocol Emulation Testbed". In: *Proceedings of the Wireless Communications and Networking Conference (WCNC)*. IEEE, Mar. 2017. DOI: 10.1109/wcnc.2017.7925815.

[122] N. Nasreddine, J. L. Boizard, C. Escriba, and J. Y. Fourniols: "Wireless Sensors Networks Emulator Implemented on a FPGA". In: *Proceedings of the 9th International Conference on Field-Programmable Technology (FPT)*. IEEE, 2010, pp. 279–282. DOI: 10.1109/FPT.2010.5681484.

[123] F. V. Gallego, J. Alonso-Zarate, C. Verikoukis, and L. Alonso: "A Survey on Prototyping Platforms for the Development and Experimental Evaluation of Medium Access Control Protocols". In: *IEEE Wireless Communications* 19.1 (Feb. 2012), pp. 74–81. DOI: 10.1109/mwc.2012.6155879.

[124] L. Ding et al.: "High Fidelity Wireless Network Evaluation for Heterogeneous Cognitive Radio and Networks". In: *Proceedings of the SPIE Defense, Security, and Sensing Conference*. May 2012. DOI: 10.1117/12.919273.

[125] O. Karfich, F. Bartols, T. Steinbach, F. Korf, and T. C. Schmidt: "Poster Abstract: A Hardware/Software Platform for Real-time Ethernet Cluster Simulation in OMNeT++". In: *Proceedings of the 6th International Workshop on OMNeT++ (OMNeT++)*. ICST. Cannes, Frankreich, Mar. 2013. DOI: 10.4108/ICST.SIMUTOOLS.2013.251698.

[126] C. Sommer, R. German, and F. Dressler: "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis". In: *IEEE Transactions on Mobile Computing* 10.1 (Jan. 2011), pp. 3–15. DOI: 10.1109/TMC.2010.133.

[127] T. Baumgartner et al.: "Demo Abstract: Bridging the Gap between Simulated Sensor Nodes and the Real World." In: *Proceedings of the 4th International Workshop on Real-World Wireless Sensor Networks (REALWSN)*. Springer. 2010, pp. 174–177.

[128] J. Kölsch, C. Heinz, S. Schumb, and C. Grimm: "Hardware-in-the-loop Simulation for Internet of Things Scenarios". In: *Proceedings of the 2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 2018, pp. 1–6. DOI: 10.1109/MSCPES.2018.8405399.

[129] S. Jafer, Q. Liu, and G. Wainer: "Synchronization Methods in Parallel and Distributed Discrete-Event Simulation". In: *Simulation Modelling Practice and Theory* 30 (Jan. 2013), pp. 54–73. DOI: 10.1016/j.simpat.2012.08.003.

[130] S. Böhm: "IEEE 802.15.4 Sensornetz-Emulation am Praxisbeispiel RoSeNet". MA thesis. Brandenburgische Technische Universität Cottbus-Senftenberg, 2014.

[131] A. Sobeih et al.: "J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks". In: *IEEE Wireless Communications magazine* 13 (2005), p. 2006.

[132] D. S. Buse et al.: "Bridging Worlds: Integrating Hardware-in-the-Loop Testing with Large-Scale VANET Simulation". In: *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, Feb. 2018. DOI: 10.23919/wons.2018.8311659.

[133] Z. Y. Song et al.: "Hy-Sim: Model Based Hybrid Simulation Framework for WSN Application Development". In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2010, p. 87. DOI: 10.4108/icst.simutools2010.8662.

[134] M. M. R. Mozumdar, L. Lavagno, L. Vanzago, and A. L. Sangiovanni-Vincentelli: "HILAC: A Framework for Hardware in the loop Simulation and Multi-Platform Automatic Code Generation of WSN Applications". In: *Proceedings of the 9th International Symposium on Industrial Embedded Systems (SIES)*. IEEE. 2010, pp. 88–97. DOI: 10.1109/sies.2010.5551370.

[135] I.-G. Lee, S.-B. Lee, and S.-C. Park: "Effective Co-Verification of IEEE 802.11a MAC/PHY Combining Emulation and Simulation Technology". In: *Proceedings of the 38th Annual Simulation Symposium (ANSS)*. IEEE, 2005. DOI: 10.1109/anss.2005.19.

[136] M. Jung and A. Hergenröder: "OMNeTA: A Hybrid Simulator for a Realistic Evaluation of Heterogeneous Networks". In: *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. Q2SWinet '15. Cancun, Mexico: Association for Computing Machinery, 2015, pp. 75–82. ISBN: 9781450337571. DOI: 10.1145/2815317.2815331.

[137] A. Kato, M. Takai, and S. Ishihara: "WiNE-Tap: Wireless LAN Emulator with Wireless Network TAP Devices". In: *Ad Hoc Networks* 123 (Dec. 2021), p. 102690. DOI: 10.1016/j.adhoc.2021.102690.

[138] T. Staub, R. Gantenbein, and T. Braun: "VirtualMesh: An Emulation Framework for Wireless Mesh and Ad Hoc Networks in OMNeT++". In: *Simulation* (2010). DOI: 10.1177/0037549710373909.

[139] L. Riliskis and E. Osipov: "Symphony: Simulation, emulation, and virtualization framework for accurate wsn experimentation". In: *Proceedings of the 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)*. IEEE. 2013, pp. 1–6. DOI: 10.1109/sesena.2013.6612257.

[140] E. Weingärtner, H. vom Lehn, and K. Wehrle: "Device Driver-enabled Wireless Network Emulation". In: *Proceedings of the 4th International Conference on Simulation Tools and Techniques (SimuTools)*. Barcelona, Spanien: ICST, Mar. 2011. DOI: 10.4108/icst.simutools.2011.245543.

[141] S. Unterschütz and V. Turau: "A Hybrid Testbed for a Seamless Combination of Wireless Sensor Networks and OMNeT++ Simulations". In: *11. GI/ITG KuVS Fachgespräch Sensornetzwerke*. GI/ITG. Darmstadt, Deutschland, Sept. 2012, pp. 52–55.

[142] J. Zhang et al.: "A Software-Hardware Emulator for Sensor Networks". In: *Proceedings of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. IEEE. Salt Lake City, UT, USA: IEEE Press, Apr. 2011, pp. 440–448. DOI: 10.1109/SAHCN.2011.5984928.

[143] P. Wehner and D. Göhringer: "Internet of Things Simulation using OMNeT++ and Hardware in the Loop". In: *Components and Services for IoT Platforms*. Springer, 2017, pp. 77–87. DOI: 10.1007/978-3-319-42304-3_4.

[144] E. Weingärtner, F. Schmidt, H. vom Lehn, T. Heer, and K. Wehrle: "SliceTime: A Platform for Scalable and Accurate Network Emulation". In: *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, MA, USA: USENIX Association, Mar. 2011.

[145] C. Obermaier, R. Riebl, and C. Facchi: "Fully Reactive Hardware-in-the-Loop Simulation for VANET Devices". In: *Proceedings of the 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Nov. 2018. DOI: 10.1109/itsc.2018.8569663.

[146] F. Klingler, G. S. Pannu, C. Sommer, and F. Dressler: "Poster: Connecting Simulation and Real World: IEEE 802.11p in the Loop". In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom)*. ACM, Oct. 2017. DOI: 10.1145/3117811.3131265.

[147] Y. Wen, W. Zhang, R. Wolski, and N. Chohan: "Simulation-based Augmented Reality for Sensor Network Development". In: *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM. 2007, pp. 275–288. DOI: 10.1145/1322263.1322290.

[148] V. Venkataramanan, P. Wang, A. Srivastava, A. Hahn, and M. Govindarasu: "Interfacing Techniques in Testbed for Cyber-Physical Security Analysis of the Electric Power Grid". In: *Proceedings of the Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, Apr. 2017. DOI: 10.1109/mscpes.2017.8064543.

[149] E. Weingärtner, F. Schmidt, T. Heer, and K. Wehrle: "Synchronized Network Emulation: Matching Prototypes with Complex Simulations". In: *ACM SIGMETRICS Performance Evaluation Review* 36.2 (Aug. 2008), pp. 58–63. DOI: 10.1145/1453175.1453185.

[150] S. Pandit and G. Singh: "Framework for Cross-Layer Optimization in Cognitive Radio Network". In: *Spectrum Sharing in Cognitive Radio Networks*. Springer International Publishing, 2017, pp. 225–251. DOI: 10.1007/978-3-319-53147-2_10.

[151] C. K. Wu et al.: "Critical Internet of Things: An Interworking Solution to Improve Service Reliability". In: *IEEE Communications Magazine* 58.1 (Jan. 2020), pp. 74–79. DOI: 10.1109/mcom.001.1900526.

[152] P. Sethi and S. R. Sarangi: "Internet of Things: Architectures, Protocols, and Applications". In: *Journal of Electrical and Computer Engineering* 2017 (2017), pp. 1–25. DOI: 10.1155/2017/9324035.

[153] H. A. Khattak, M. A. Shah, S. Khan, I. Ali, and M. Imran: "Perception layer security in Internet of Things". In: *Future Generation Computer Systems* 100 (Nov. 2019), pp. 144–164. DOI: 10.1016/j.future.2019.04.038.

[154] T. O'Shea and J. Hoydis: "An Introduction to Deep Learning for the Physical Layer". In: *IEEE Transactions on Cognitive Communications and Networking* 3.4 (Dec. 2017), pp. 563–575. DOI: 10.1109/tccn.2017.2758370.

[155] C. Jiang et al.: "Machine Learning Paradigms for Next-Generation Wireless Networks". In: *IEEE Wireless Communications* 24.2 (Apr. 2017), pp. 98–105. DOI: 10.1109/mwc.2016.1500356wc.

[156] Z. Qin, H. Ye, G. Y. Li, and B.-H. F. Juang: "Deep Learning in Physical Layer Communications". In: *IEEE Wireless Communications* 26.2 (Apr. 2019), pp. 93–99. DOI: 10.1109/mwc.2019.1800601.

[157] S. S. Wagh, A. More, and P. R. Kharote: "Performance Evaluation of IEEE 802.15.4 Protocol Under Coexistence of WiFi 802.11b". In: *Procedia Computer Science* 57 (2015), pp. 745–751. DOI: 10.1016/j.procs.2015.07.467.

[158] M. Tüxen et al.: *PCAP Next Generation (pcapng) Capture File Format*. Internet-Draft draft-tuexen-opsawg-pcapng-04. Work in Progress. Internet Engineering Task Force, Oct. 2021. 57 pp. URL: https://datatracker.ietf.org/doc/html/draft-tuexen-opsawg-pcapng-04.

[159] F. Javed, M. K. Afzal, M. Sharif, and B.-S. Kim: "Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review". In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 2062–2100. DOI: 10.1109/comst.2018.2817685.

[160] IEEE Standards Association: *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std 1588. The Institute of Electrical and Electronics Engineers, Inc., 2020. DOI: 10.1109/IEEESTD.2020.9120376.

[161] B. Bloessl, C. Leitner, F. Dressler, and C. Sommer: "A GNU Radio-based IEEE 802.15.4 Testbed". In: *12. GI/ITG KuVS Fachgespräch Sensornetzwerke*. Ed. by J. Nolte. GI/ITG. BTU Cottbus, Institut für Informatik, Sept. 2013, pp. 37–40.

[162] F. Wunsch, K. Maier, H. Jäkel, and F. K. Jondral: "Implementation and Performance Evaluation of IEEE 802.15.4 LECIM DSSS PHY at 2.4 GHz". In: *Proceedings of the 7th GNU Radio Conference, San Diego, CA, September 11-15, 2017*. 2017, p. 6.

[163] R. Zitouni, S. Ataman, M. Mathian, and L. George: "IEEE 802.15.4 Transceiver for the 868/915 MHz Band using Software Defined Radio". In: *Proceedings of SDR'12-WInnComm-Europe, 27-29 June 2012* (Apr. 30, 2013). arXiv: 1304.8028 [cs.NI].

[164] L. Abeni and D. Faggioli: "Using Xen and KVM as Real-Time Hypervisors". In: *Journal of Systems Architecture* 106 (June 2020), p. 101709. DOI: 10.1016/j.sysarc.2020.101709.

[165] S. Böhm: "Hybrid IEEE 802.15.4 Network Emulation". In: *Proceedings of the 2015 Networked Systems (NetSys) PhD Forum*. Poster Abstract. Mar. 2015. DOI: 10.1109/NetSys34399.2015.

[166] K. B. Rasmussen and S. Capkun: "Implications of Radio Fingerprinting on the Security of Sensor Networks". In: *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*. IEEE, 2007. DOI: 10.1109/seccom.2007.4550352.

[167] Y. Wu et al.: "Network Planning in Wireless Ad Hoc Networks: A Cross-Layer Approach". In: *IEEE Journal on Selected Areas in Communications* 23.1 (Jan. 2005), pp. 136–150. DOI: 10.1109/jsac.2004.837362.

[168] B. Ousat and M. Ghaderi: "LoRa Network Planning: Gateway Placement and Device Configuration". In: *Proceedings of the 2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, July 2019. DOI: 10.1109/iciot.2019.00017.

[169] S. Ivanov and E. Nett: "Achieving Fault-Tolerant Network Topology in Wireless Mesh Networks". In: *Wireless Mesh Networks - Efficient Link Scheduling, Channel Assignment and Network Planning Strategies*. InTech, Aug. 2012. DOI: 10.5772/50173.

[170] E. H. Houssein et al.: "Optimal Sink Node Placement in Large Scale Wireless Sensor Networks Based on Harris' Hawk Optimization Algorithm". In: *IEEE Access* 8 (2020), pp. 19381–19397. DOI: 10.1109/access.2020.2968981.

[171] K. Akkaya and M. Younis: "Relocation of Gateway for Wnhanced Timeliness in Wireless Sensor Networks". In: *Proceedings of the IEEE International Conference on Performance, Computing, and Communications, 2004*. IEEE. DOI: 10.1109/pccc.2004.1395064.

[172] H. Breu and D. G. Kirkpatrick: "Unit disk graph recognition is NP-hard". In: *Computational Geometry* 9.1-2 (Jan. 1998), pp. 3–24. DOI: 10.1016/s0925-7721(97)00014-x.

[173] G. Judd and P. Steenkiste: "Using Emulation to Understand and Improve Wireless Networks and Applications". In: *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)*. Berkeley, CA, USA: USENIX Association, 2005, pp. 203–216. URL: http://dl.acm.org/citation.cfm?id=1251203.1251218.

[174] X. E. Li: *Channel Identification and Equalization in Digital Communications*. Binghamton University, State University of New York, June 2022. URL: http://www.ws.binghamton.edu/li/tutor/channel.html.

[175] J. Morman: "Channel Equalization using GNU Radio". In: *Proceedings of the 2020 Free and Open source Software Developers' European Meeting (FOSDEM)*. 2020. URL: https://archive.fosdem.org/2020/schedule/event/fsr_hannel_equalization_using_gnu_radio/attachments/slides/3907/export/events/attachments/fsr_hannel_equalization_using_gnu_radio/slides/3907/equalizers_fosdem_2020.pdf.

[176] M. Elmer, W. Schaaf, D. Biemelt, W. Gerwin, and R. F. Hüttl: *The artificial catchment 'Chicken Creek' - initial ecosystem development 2005-2010*. Tech. rep. 3. Forschungszentrum Landschaftsentwicklung und Bergbaulandschaften (FZLB), 2012.

[177] S. Mehner: "Hühnerwasser goes smart! Ein Konzept für ein nachhaltiges Umweltmonitoringsystem". MA thesis. Brandenburgische Technische Universität Cottbus-Senftenberg, 2015.

[178] D. G. Andersen: "Theoretical approaches to node assignment". In: *Computer Science Department* (2002), p. 86.

[179] R. Ricci, C. Alfeld, and J. Lepreau: "A Solver for the Network Testbed Mapping Problem". In: *ACM SIGCOMM Computer Communication Review* 33.2 (Apr. 2003), pp. 65–81. DOI: 10.1145/956981.956988.

[180] R. McGeer, D. G. Andersen, and S. Schwab: "The Network Testbed Mapping Problem". In: *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, 2011, pp. 383–398. DOI: 10.1007/978-3-642-17851-1_30.

[181] G. D. da Fonseca, V. G. P. de Sá, R. C. S. Machado, and C. M. H. de Figueiredo: "On the Recognition of Unit Disk Graphs and the Distance Geometry Problem with Ranges". In: *Discrete Applied Mathematics* 197 (2015), pp. 3–19.

[182] M. D. Francesco, G. Anastasi, M. Conti, S. K. Das, and V. Neri: "Reliability and Energy-Efficiency in IEEE 802.15.4/ZigBee Sensor Networks: An Adaptive and Cross-Layer Approach". In: *IEEE Journal on Selected Areas in Communications* 29.8 (Sept. 2011), pp. 1508–1524. DOI: 10.1109/jsac.2011.110902.

[183] F. Cuomo, A. Abbagnale, and E. Cipollone: "Cross-Layer Network Formation for Energy-Efficient IEEE 802.15.4/ZigBee Wireless Sensor Networks". In: *Ad Hoc Networks* 11.2 (Mar. 2013), pp. 672–686. DOI: 10.1016/j.adhoc.2011.11.006.

[184] M. Al-Jemeli and F. A. Hussin: "An Energy Efficient Cross-Layer Network Operation Model for IEEE 802.15.4-Based Mobile Wireless Sensor Networks". In: *IEEE Sensors Journal* 15.2 (Feb. 2015), pp. 684–692. DOI: 10.1109/jsen.2014.2352041.

[185] P. Detterer, M. Nabi, H. Jiao, and T. Basten: "Receiver-Sensitivity Control for Energy-Efficient IoT Networks". In: *IEEE Communications Letters* 25.4 (Apr. 2021), pp. 1383–1386. DOI: 10.1109/lcomm.2020.3041935.

[186] G. Glodni: "Cross-Layer WSN Simulation mittels Software-Defined-Radio (SDR)". MA thesis. Brandenburgische Technische Universität Cottbus-Senftenberg, 2019.

[187] J. Ren et al.: "RF Energy Harvesting and Transfer in Cognitive Radio Sensor Networks: Opportunities and Challenges". In: *IEEE Communications Magazine* 56.1 (Jan. 2018), pp. 104–110. DOI: 10.1109/mcom.2018.1700519.

[188] R.-F. Liao et al.: "Deep-Learning-based Physical Layer Authentication for Industrial Wireless Sensor Networks". In: *sensors* 19.11 (2019), p. 2440.

[189] H. Kim, S. Oh, and P. Viswanath: "Physical Layer Communication via Deep Learning". In: *IEEE Journal on Selected Areas in Information Theory* 1.1 (May 2020), pp. 5–18. DOI: 10.1109/jsait.2020.2991562.

[190] G. F. Riley and T. R. Henderson: "The ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Ed. by K. Wehrle, M. Guenes, and J. Gross. Springer, Mar. 2010. Chap. 2, pp. 15–34. DOI: 10.1007/978-3-642-12331-3_2.

[191] U. Lamping, R. Sharpe, and E. Warnicke: *Wireshark Users Guide*. Oct. 2021. URL: http://www.wireshark.org/docs/wsug_html_chunked/.

Availability of online materials and web links were last checked in April 2022.