Modular Energy-Aware Simulation Environment for Wireless Sensor Network Evaluation

Bachelor-Thesis

Lukas Wehrstein
KOM-B-0710

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Modular Energy-Aware Simulation Environment for Wireless Sensor Network Evaluation
Modulare Energiesensitive Simulationsumgebung zur Evaluation Drahtloser Sensornetzwerke

Bachelor-Thesis
Studiengang: Elektrotechnik und Informationstechnik
KOM-B-0710

Eingereicht von Lukas Wehrstein
Tag der Einreichung: 11. April 2022

Gutachter: Prof. Dr.-Ing. Ralf Steinmetz
Betreuer: Julian Zobel

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation (KOM)
Prof. Dr.-Ing. Ralf Steinmetz

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Lukas Wehrstein, die vorliegende Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Darmstadt, den 11. April 2022

_____

Lukas Wehrstein

Contents

Abstract

Wireless Sensor Networks (WSNs) are part of the *Internet of Things* and are particularly useful for analysing data from physically distributed sensors in remote locations. Most of these sensors are battery-powered which together with the wireless data transmission makes them very flexible and mobile. Thus, there is a wide range of different applications for WSNs. Due to the energy supply of the sensor nodes by batteries, there is a strong dependence of the network on the energy-related lifetime of the individual sensor nodes. To be able to evaluate new possibilities and their applicability in advance, simulations are necessary.

In this work, a model is presented that enables the simulation of sensor nodes with modular structure concerning their energy-related lifetime. The modular approach is particularly interesting, as it allows sensor nodes to be quickly and flexibly assembled from various components. For the simulation, sensor nodes are therefore modelled as individual components. This enormously simplifies the process of simulation configuration since consumption data for individual components are partly available in datasheets or can easily be determined. In addition, already modelled components can be reused for further sensor node configurations.

The model presented in this work is implemented in the *Simonstrator* network simulator. Related to real-world measurements the simulation using a highly simplified battery model achieves an accuracy with an average deviation of 30 %. Using non-linear extended battery models, an improvement of the accuracy down to 5 % deviation would be possible. Thus, the model shown in this work enables the simulation of individually configured modular sensor nodes in WSNs with regard to their energy-related lifetime.

1 Introduction

This work presents a model for modular energy-aware simulation for Wireless Sensor Networks (WSNs). These WSNs are systems that enable the wireless collection and aggregation of sensor data from distributed sources. They consist of multiple sensor nodes, that are mostly self-powered and often distributed over large distances of sometimes up to several kilometres. From there, the individual sensor nodes are able to communicate using wireless transmission technologies like LoRa, Sigfox, WiFi and others. As these sensor nodes are often battery-powered, they are very versatile and can be used in remote areas without electricity infrastructure. Accordingly, components that consume little energy are usually installed in such sensor nodes, so that the lifetime of the sensor nodes can be increased. In this work, a model is presented that enables modular energy-aware simulation for flexible sensor node configurations.

The fact that the sensor nodes in a WSN are autonomous, brings many advantages, such as that they can be used flexible and everywhere, independent of the given infrastructure. But on the other hand, these nodes are thus strongly dependent on their energy supply, which in most cases is implemented via a battery. Even if these sensor nodes are designed to have low energy consumption, the batteries discharge over time. Based on the application and the connected hardware, sensor nodes can reach up to several years of battery life. In some WSN applications, energy harvesting modules like solar panels or small wind turbines are used to recharge the batteries [AAT+18, HGL+21]. But in general, there is a great dependency of a WSN on the batteries and energy components of the individual sensor nodes. Therefore, after a certain time, sensor nodes within a WSN may no longer work properly, due to energy deficiency. This means that the central sensor data collection unit, for example a server, is no longer supplied with new sensor data.

Depending on the application, missing out on data or complete network failures due to discharged sensor nodes can cause great damage or even endanger human life. To prevent this, the batteries of the sensor nodes must be replaced or recharged regularly. Since these sensor nodes can also be used in rough terrain, the replacement of batteries in the sensor nodes is elaborate and charges high costs, especially when WSNs consist of a high number of distributed sensor nodes.

Therefore, it is important, both economically and to ensure the function of the network, to know as precisely as possible how long the battery lifetime is for individual sensor nodes. With this knowledge, batteries can then be replaced in time without causing energy-related sensor node failures.

In addition to live operation, where the current battery status could at least be evaluated via battery sensors, it is also interesting for the development and design of WSNs to be able to know the energy-related lifetimes of individual sensor nodes as precisely as possible. This also enables an improvement in the development process for devices in WSNs [MWHA09].

To enable the simulation to easily cover as many different scenarios as possible, the goal of this work is to develop a model, that supports the energy-aware simulation for modular configured sensor nodes. This means that the sensor nodes can be composed of several different components with a specified or configurable energy consumption. Further, the division of a sensor node into individual components also has the advantage that components or hardware modules that have already been specified can be reused for other configurations. This enables fast use of energy-aware simulations for a wide range of different WSN applications and beyond.

1.1 Problem Statement and Contribution

There are already simulators that are suitable or even designed for the simulation of WSNs. However, the focus is often on the simulation of data traffic within the network. Since the energy state of the sensor nodes has an impact on the functionality of the WSN, the consideration of energy consumption

in the sensor nodes is essential for a good WSN simulation. To simulate the energy consumption of sensor nodes, they must first be analysed in terms of their consumption. This is elaborate and requires measurements with a prototype sensor node.

Therefore, in this work, a modular approach for simulating sensor node lifetimes based on the individual components of a sensor node is developed. This simplifies the simulation configuration, as sensor nodes are a composition of several components with respective energy consumption. Instead of analysing the whole node, information about the used components is sufficient to configure the here presented model. Based on the consumption of the individual components, which may be provided by the corresponding manufacturers, the model creates an energy model for the entire sensor node, which is then used for the simulation of sensor node lifetimes. The modular structure of the model also supports the implementation and configuration of any sensor node workloads. This means that components of the sensor nodes can be dynamically switched on and off during the simulation depending on the task and state of the sensor node's workload, which is essential for energy-based lifetime simulation.

For this approach, this work presents a conceptual design of this modular sensor node model. Further, a sample implementation of this design is implemented in the *Simonstrator* [RSRS15] simulation framework. A *proof-of-concept* evaluation of this implementation is also done. For this purpose, the simulated lifetimes are also compared to real-world measurements and evaluated with regard to the precision of the simulation model. It turns out that this model is quite suitable for the lifetime simulation of sensor nodes in WSNs. However, the precision of the simulation depends significantly on the quality of the battery model used. Using a simple ideal linear battery model, the simulation is only able to deliver results with a deviation of about 30% from the real-world measurement. With more accurate battery models, on the other hand, results with up to 5% deviation are possible.

1.2 Outline

This work is structured in several chapters. Chapter 2 gives background information on which this work is based. Related work and work that affects parts of this thesis is explained in Chapter 3. Chapter 4 then presents the design developed in this work. An exemplary implementation of this design is then shown in Chapter 5. Chapter 6 provides an evaluation and analysis of the approach. Finally, Chapter 7 summarises the achievements of this work and provides an outlook for potential future work based on this work.

2 Background

This chapter gives information about the background of this work. In Section 2.1, requirements and the typical structure of Wireless Sensor Networks (WSNs) are described. Common areas of application for WSNs are shown in Section 2.2. As specific transmission technologies are key elements in WSNs, different typical WSN transmission technologies are presented and compared in Section 2.3. Finally, Section 2.4 summarizes the background information given in this chapter.

2.1 Requirements and Structure of Wireless Sensor Networks

Since the spread of the internet, it has been possible to transmit data easily and worldwide for example to control and monitor other devices and systems remotely. Internet of Things (IoT) therefore means the networking of a wide variety of devices and is used for example in the industrial, research and consumer sectors. For this purpose, over 26 billion IoT devices were active in 2020 and 75 billion connected IoT devices are estimated for 2025 [Lou21]. Having access to devices and information from all over the world, some even with real-time access makes the field of application for IoT large and inhomogeneous. These reach from smart homes in the consumer segment to global climate surveillance with distributed sensor networks.

WSNs are a sub-area of IoT where the main purpose is to enable sensors that are physically distributed over large distances to communicate with each other and wirelessly transmit their collected data to a central remote server unit. This makes it possible to access, analyse and react to various sensor data fully automatically and from anywhere. Therefore, WSNs are also widely used in agriculture and industry to further automate processes and thus increase for example the efficiency level and safety of production.

On the other hand, the use of WSNs further increases the dependence on the global network infrastructure and generated data traffic and server applications increase the global power consumption. However, depending on the application, the use of IoT or WSNs enables the power consumption for example in industrial plants can be reduced, due to further optimised processes. Especially the feature of complete wireless sensor nodes with low energy consumption, long transmission ranges and battery lifetimes of several months up to years, enables many new fields of application and promotes innovation [KPKK19].

To enable remote access on nodes in a WSN, most applications rely on the same concept. As illustrated in Figure 2.1, the key element of a WSN is the wireless transmission technology used for communication. Thereby, the used technology is mostly optimized for the application needs of low power and long-range transmissions. However, this also means, that the available data rate is mostly in the kilobit area. Since every sensor node is equipped with a wireless transceiver, the sensor nodes are able to communicate with each other. This is required for example for connected smoke detectors or other applications with a decentralized approach. For applications that require central access to the data from all nodes, mostly a gateway is used to transfer data from the local wireless network via these special low power long-range transmission technologies to a server on the internet. Such gateways can be developed by the user himself, but there are also companies and organisations that provide a gateway infrastructure for a specific transmission technology. The availability of such gateway providers differs greatly depending on the location. Instead of building many new gateways and connecting them to the internet, some technologies are able to use and adapt the existing mobile radio infrastructure. However, this option is usually associated with higher costs and shorter battery lifetimes [MBCM18]. A basic structure of a WSN with multiple sensors nodes and a server with a connected gateway is shown in Figure 2.1.

Even if the communication directly from the sensor node to the server is usually used, it is also possible to set up communication between the nodes themselves and from the server to the nodes. However, due to more frequent wake-up times in order to be able to receive incoming messages, the energy consumption of the sensor nodes also increases.

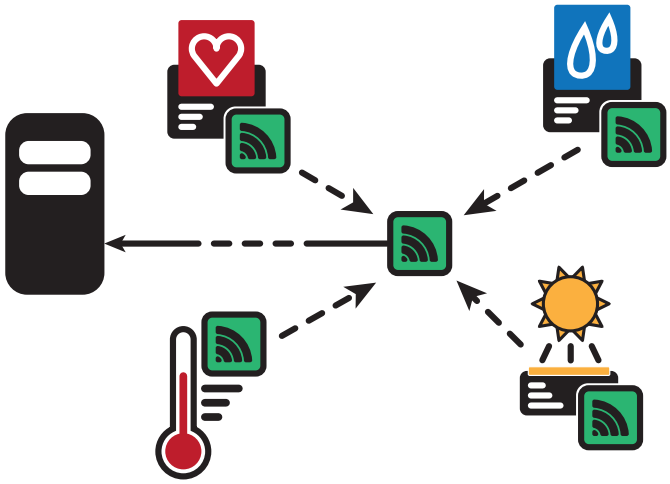


Figure 2.1: General structure of a unidirectional WSN. Several sensor nodes collect and send data to a central server via a wireless transmission medium. To enable the server to receive the data from the sensors a gateway is used to forward data from the wireless medium to the server / local computer.

2.2 Applications of Wireless Sensor Networks

WSNs respective the corresponding individual sensor nodes are able to be used in any conceivable environment, for example in rough terrain, areas without energy infrastructure, buildings, industrial plants, hospitals, disaster areas, forests, fields or even in exceptional cases like underwater or in space [AA11, AAIS14, KGT09, ASM+11, RAA+14]. Therefore, WSNs are suitable solutions for IoT applications for indoor and outdoor environmental monitoring, agriculture, logistics, health, smart home, smart cities, military, security or robotics [ALM05, KNVK20]. This usually involves making sensor data such as weather, position, heart frequency, or electricity measurements from different locations accessible for monitoring and optimisation.

While in some systems a failure of single sensor nodes is not particularly tragic, for other applications it is important that every node functions reliably and fail-safe. Especially in health-care applications for real-time monitoring of e.g. pulse and blood oxygen, failures of such a WSN system can be even critical for survival [KTK15, KGT09]. It is therefore important that sensor nodes can be analysed and observed in terms of their lifetime. This also shows how the specific application of a WSN has a great impact on the specific implementation of the WSN. Especially when selecting a suitable transmission technology for the sensor nodes and the gateways, there are major differences between the various options.

2.3 Wireless Transmission Technologies for Wireless Sensor Network

There are multiple different wireless transmission technologies, which are commonly used for WSN applications. Depending on the specific structure and requirements of the application, different technologies are best suited. In general, these can be specified in terms of transmission range, data rate, cost, energy consumption and quality of service. However, these specifications are often not independent of each other. For example, a long transmission range with low power consumption automatically implies a lower data rate. On the other hand, other technologies provide higher data rates but with shorter transmission ranges or higher energy consumption. Figure 2.2 gives an overview of the transmission range and energy consumption of different wireless transmission technologies. LoRa and Sigfox in particular stand out, as they provide long transmission ranges of up to several kilometres and low power consumption. However, the proprietary technology by Sigfox can only be used with their associated network and gateway infrastructure. LoRa is also a proprietary modulation technique, but it can be used either in its own local networks or within a global network using the open LoRaWAN protocol,

which is developed and specified by the LoRa Alliance organisation. On the downside, the data rates of LoRaWAN and Sigfox are very restricted to $0.3 - 50$ kbit and 0.1 kbit. Other technologies like WiFi and cellular approaches are suitable for applications that do not require these extreme low power consumption and long ranges, but a higher data rate. Even if these provide low energy consumption, NFC and other RFID technologies and Bluetooth are typically not used for WSNs because their transmission range is too short for taking advantage of wireless transmission.

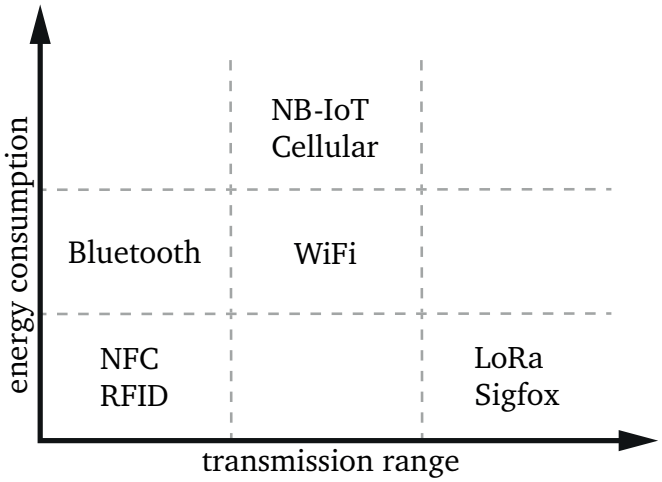


Figure 2.2: Qualitative classification of different wireless transmission technologies in terms of transmission range and energy consumption.

Further, there are generally two options for WSNs in terms of the network structure. There are global networks like Sigfox, LoRaWAN or the cellular network which can be used as gateways, eventually making the data available all over the world. Another option is to set up an own local network with an own gateway and sort of local server, to save and proceed the transmitted data. With some technologies it is also possible to build a mixture of local and global networks. The Things Network (TTN) for example gives the opportunity to add own LoRaWAN gateways to their public network.

Overall, both in terms of power consumption and due to the different ranges and network topologies, transmission technology is a decisive factor in designing a WSN. It is therefore important that a simulation takes different transmission technologies into account and delivers results that are as accurate as possible.

2.4 Summary

This chapter gives background information that further explains the structure and applications of WSNs. These represent a sub-area of the field IoT. With the spread of the internet and the availability of various low power, long-range transmission technologies, WSNs are currently popular and a growing sector. Depending on the specific application, the implementation of WSNs may differ, however, the base structure mostly remains the same. Sensors and other hardware modules are equipped with a transmission module that is able to communicate over a certain distance with other sensor nodes or a gateway. Gateways are interfaces that provide a transfer from the local wireless network to the internet. The use of such gateways in combination with long-range transmission technologies makes it possible to cover large areas and forward data from distributed wireless sensor nodes to a server on the internet. This makes WSNs especially interesting for areas with no electrical infrastructure. Different wireless transmission technologies and networks come with different specifications, which are suitable for different WSN scenarios, depending on their actual application.

3 Related Work

There is already work that addresses specific components of this work. Section 3.1 is split into five subsections and discusses what progress has already been made in the field of energy models for batteries and Wireless Sensor Network (WSN)-node components, how a battery lifetime prediction can be made from this and how WSN modules can be analysed in terms of measuring power consumption. Section 3.2 examines the structure of existing energy-aware simulations. In Section 3.3 an analysis of the related work is done. Finally, Section 3.4 summarizes this chapter.

3.1 Energy Models and Battery Lifetime Prediction

Sensor nodes in WSNs are mostly battery-based. This means, even with connected energy harvesting modules the available energy of a wireless sensor node is limited. Since WSN nodes typically contain one energy store like a battery and several energy consumers like Microcontrollers (MCs), radio modules or sensors the precision of the battery model is one of the most important factors to achieve realistically energy-based lifetime predictions [RMB+17, KCZ07].

3.1.1 Influence of Non-Ideal Battery Behaviour

The accuracy of the simulated lifetime depends heavily on the battery model, as it plays a central role in the sensor node [RSKN05]. Unlike an ideal battery, real batteries do not behave in a linear and ideal way. Common non-idealities are Rate-Capacity-Effect, Recovery-Effect, temperature, cycle number and self-discharge.

The Rate-Capacity-Effect, based on Peukert's Law [Peu97], describes a non-linear behaviour in batteries in which the maximum usable energy from the battery is lower at high current drains than at lower current drains. The Recovery-Effect describes the partially increasing of the battery output voltage when the battery is not in use.

Due to different types of batteries using different chemical elements and technologies, the influence of these parameters on the battery behaviour varies. According to Chen et al. [CR06] the most important effects on low power polymer Li-ion batteries are the *Rate-Capacity-Effect* and the *Recovery-Effect*, which is why the authors neglect the other effects named above. On the other hand Kerasiotis et al. [KPA+10] also neglects the Recovery-Effect for WSN scenarios, where the current drains are comparatively low. There are also further works about other non-ideal battery characteristics [KK16, HD13, OROR18].

3.1.2 Battery Models

The most common models for non-ideal batteries typically used in Embedded Systems are the *Kinetic Battery Model (KiBaM)* and the *Stochastic Battery Model (StBaM)*.

The KiBaM represents a battery as two wells of charges: the *available-charge* well and the *bound-charge* well. Charges from the *bound-charge* well only flow to the *available-charge* well. Representing the current drain, charges from the *available-charge* well flow to the load. After taking charges out of the *available-charge* well, a charge current from the *bound-charge* balances the two wells again. Referring to Rao et al. [RSKN05], the KiBaM is well suited to understand the Recovery-Effect but needs a few modifications for battery types used for mobile applications.

The StBaM is based on a Markov process. Using several parameters and probabilities, the StBaM defines different states and transitions. The behaviour of the battery can be analysed by observing the

maximal-available capacity T and the *nominal capacity* N. T defines the absolute maximum of energy a load can obtain from the battery. N is the actual available capacity. Due to the Recovery-Effect, the battery's *nominal capacity* N may increase in non-discharging idle states. The battery is considered to be fully discharged if the *maximal-available capacity* T is depleted or if the *nominal capacity* N equals zero [RSKN05].

An in-depth look and comparison of different battery models is provided by Kaj and Konané [KK16]. There is also a variety of different extended, adapted and combined battery models [RSKN05, ORoR18, KPMB08, HD13].

3.1.3 Battery Lifetime Prediction

There are different approaches to battery lifetime prediction. Wang et al. [WXVS14] develop a model for reliability and lifetime prediction of wireless sensor nodes. The authors equate the probability of failure of a sensor node with the depletion of the battery, which once again shows the importance of a realistic battery model for WSN-node lifetime prediction. In addition, they determine a threshold capacity C_{min}, which represents the amount of battery capacity that can not be used due to non-idealities. Using Peukert's Law [Peu97] for calculating the remaining capacity $C(t)$ in dependence of time t and a constant current charge I, they achieve a concise but Rate-Capacity-Effect-sensitive formula for the sensor node reliability $R(t)$:

$$R(t) = Pr\{C(t) \geq C_{min}\} \tag{3.1}$$

$$C(t) = C_0 - I^k \cdot t \tag{3.2}$$

where $Pr\{X\}$ returns the probability of event X, C_0 is the initial capacity of the battery and k is the Peukert constant with $k = 1$ for a linear battery model and k between 1.1 and 1.3 for different battery types.

A similar approach to modelling battery lifetime was also taken by Kerasiotis et al. [KPA+10] by considering the Rate-Capacity-Effect and neglecting the influence of temperature, self-discharge and Recovery-Effect. Using equation 3.3, Kerasiotis et al. calculate the battery lifetime t_{max} from the currents I_i and the corresponding interval durations Δt_i respectively the overall average current $I_{average}$ and the offered capacity by the battery $E_{offered}$ in mA h:

$$E_{offered} = \int_{t=0}^{t=t_{max}} I(t)dt = \sum_{i=0}^{i=N} I_i \Delta t_i = I_{average} t_{max} \Leftrightarrow t_{max} = \frac{E_{offered}}{I_{average}} \tag{3.3}$$

where N represents the number of operations with different current consumption. In their work, they also consider different duty cycles. For this purpose, Kerasiotis et al. adjust the average current accordingly and set the sleep currents to zero. Finally, with their lifetime prediction model, Kerasiotis et al. achieve an error rate of 2.7 %.

Another approach by Rao et al. [RSKN05] is to combine the KiBaM with the StBaM. With their proposed "Stochastic Modified KiBaM", Rao et al. achieve a precise battery lifetime prediction model with an error rate lower than 2.65 % for constant and pulsed current consumption. In their evaluation setup, for constant load and *high* states of pulsed loads, Rao et al. work with a current drain of 960 mA, which is set via a load resistor R_C. For the *low* phases of the pulsed load sequence, the current drain is set to 0 mA. Due to a lower current consumption per time interval, the lifetime values for pulsed loads are higher than for constant loads. But also the overall delivered charges increase with longer *low* phases. Using two 1.2 V AAA Ni-MH batteries with a charge of 1000 mA h each, the authors measured 1440 mA h (predicted: 1445.8 mA h) with a constant current load. With a pulse consisting of 2 s *high* and 3.5 s *low*, the delivered charge was 1844.5 mA h (predicted: 1830.5 mA h). However, it is questionable whether the Recovery-Effect shown here also has a comparable influence for currents significantly below 960 mA.

In another work, Barboni et al. [BV08] show the importance of precise information about cutoff voltages of the used components in the WSN node. Due to the decreasing voltage of non-ideal batteries, some hardware components will no longer function properly once the supply voltage falls below a certain cutoff. These cutoff voltages can sometimes be found in datasheets. In the author's case, these values were not reliable and the used Atmega128L operates with a voltage supply down to 2.1 V instead of 2.7 V as specified in the datasheet. This deviation causes the predicted node lifetime of 15 d to differ significantly from the actually measured lifetime of 35 d. Otherwise, the authors developed certain WSN-specific benchmarks based on which they determine the power consumption for common tasks of a WSN node. When considering the impact of transmit power on the energy consumption, Barboni et al. found that due to the other processes like MC wake-up, Radio Transceiver Wake-Up, Radio setup, calibration and listening to incoming answers, the energy savings from lower transmit powers can be neglected. The authors also neglect the power consumption of the MC while it is in deep sleep mode.

Another interesting work is done by Chen et al. [CR06]. Based on an extensive series of measurements, they set up various mathematical functions for NiMH and polymer Li-ion batteries that allow the calculation of different parameters depending on the battery State of Charge (SoC). Of particular interest is equation 3.4 for calculating the open-circuit voltage V_{OC} of a polymer Li-ion battery with a nominal output voltage of 4.1 V as a function of the SoC. Figure 3.1 shows 3.4 as a function of SoC.

$$V_{OC}(SoC) = -1.031 \cdot e^{-35 \cdot SoC} + 3.685 + 0.2156 \cdot SoC - 0.1178 \cdot SOC^2 + 0.3201 \cdot SoC^3 \tag{3.4}$$

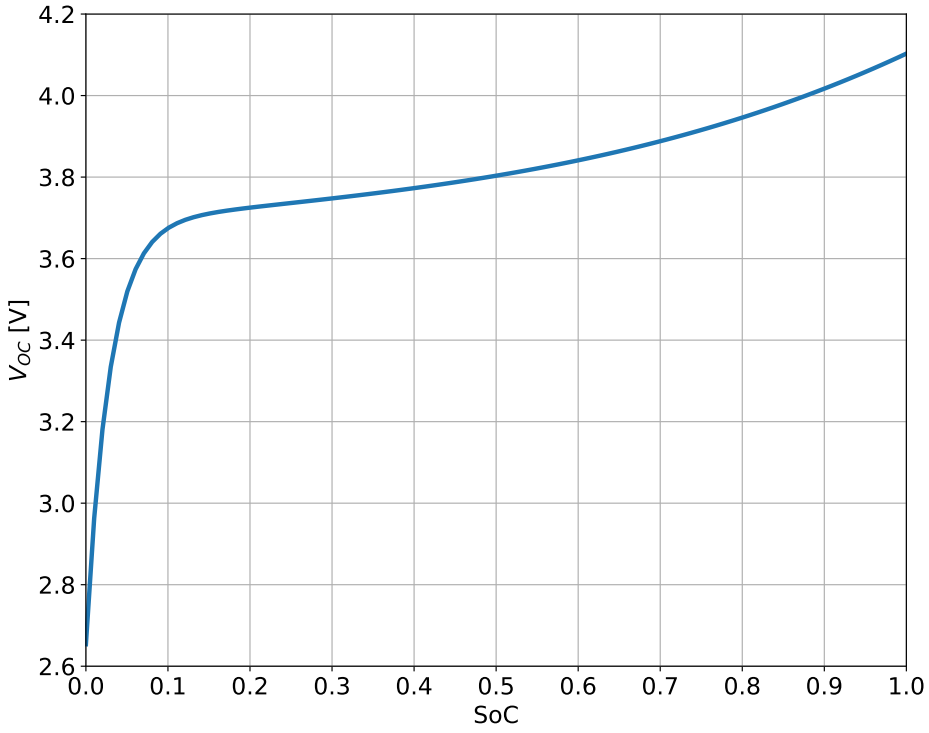


Figure 3.1: Plot of Chen et al. [CR06] proposed battery output voltage function 3.4 for a 4.1 V polymer Li-ion battery. SoC represents the available charges of the battery from 1 (fully charged) to 0 (fully discharged).

To validate the proposed mathematical model, Chen et al. use three different discharge scenarios to represent realistic scenarios. First, a constant load current of 80 mA. The second scenario consists

of different pulsed charge and discharge current profiles. The third scenario covers a periodical discharge current profile by alternating periodically between 0, 400, 160 and 640 mA. For all scenarios, the absolute voltage errors are under 30 mV and the relative battery lifetime errors are lower than 0.4 %. However, these data are only validated for a specific polymer Li-ion battery type.

3.1.4 Energy consumption of typical sensor node modules, activities and protocols

Besides the energy storage side, the consumer side of a sensor node is also important for a good energy model. Due to different hardware modules and configurations such as duty cycles and sampling rates, the individual energy consumptions of different sensor nodes may differ significantly.

Examining MCs, internal flash storage operations, timers, LEDs and a radio module Antonopoulos et al. [APS+09] measure the impact of some specific hardware components on the overall node power consumption. Using the TelosB platform, there are great differences in power consumption by switching between the six Low Power Mode (LPM)-states, including the active mode (LPM 0). In active mode, the MC consumes maximum 2.25 mA, but in the LPMs one to five, the MC reduces the current to 180 µA down to 15 µA. Operations on the internal flash-storage are mostly negligible, as they only last under 24 ms with a current consumption of 2.4 mA for writing 90 B of data. The consumption is even lower when reading data. The examined timer module is dependent on the CPU clock but has a current consumption of 90 µA to 180 µA. The LEDs come with a relatively high current consumption. Depending on the model and colour, the LEDs consume between 3.3 mA and 5.3 mA. Therefore, Antonopoulos et al. also advise against using LEDs whenever possible or letting LEDs blink instead of switching them on permanently. Finally, the radio module is the most significant one regarding power consumption with up to 20 mA. The authors investigate different modes of transmit power in terms of node energy consumption. However, since the radio module is used together with the MC, the savings from the reduced transmit power are not as high as expected compared to switching on the radio module in idle mode. Similar currents were also measured for comparable modules by [LWG05].

Based on the data from Antonopoulos et al [APS+09], Kerasiotis et al. [KPA+10] examine a scenario using the same MC, radio module, LEDs and Analog to Digital Converter (ADC). With these hardware modules, Kerasiotis et al. define four operating states with different activities in terms of radio transmission, receiving and LED state. Using the TelosB platform, with active CPU and ADC they achieve average current values between 18 mA and 23 mA for the first three basic operations with active transmission and 3.7 mA for the scenario without transmission but active LEDs. The authors also notice that down to a certain threshold voltage, the current consumption of the radio module remains rather constant, while the current consumption of the LEDs decreases proportionally to the falling voltage. Thus, different hardware components differ not only in their general power consumption but also in their current-voltage behaviour, which makes a general but precise power consumption model even more complicated.

Examining different MAC-Layer protocols (Traditional MAC, S-MAC and T-MAC), Wang et al. [WXVS14] find out that different wake-up cycles from diverse MAC protocols have a huge impact on the node's battery lifetime. Traditional MAC keeps the node in active mode all the time. S-MAC defines constant periods for sleeping and being active. Using the T-MAC protocol, the duration of active and sleep-mode are not fixed and stochastically distributed within certain limitations. T-MAC performs best with about 2.9 times longer lifetime than traditional MAC. S-Mac performs 1.8 times better than traditional MAC. Overall, Wang et al. show the importance of knowing the individual activity cycles, especially the deep sleep cycles, of the sensor nodes to accurately simulate their battery life.

3.1.5 Energy Measurement of Hardware Modules

The power consumption of a typical WSN-node differs greatly between active and sleep mode. With sleep currents lower than 100 µA, the ratio between sleep and active current can for some modules be

less than 1 %[1]. Di Nisio et al. [DNDNCS16] compare three different methods to measure currents of MC-based applications with a high dynamic range. The three methods under consideration are: shunt only, shunt with Instrumentation Amplifier (INA) and a setup with an INA with a feedback loop. In doing so, they conclude that for measurements with such a large dynamic range, the method they presented with a feedback loop is better suited than the methods with shunt only or shunt and INA. That means, that for the shunt with INA and feedback loop setup the relative error is lower, the supply voltage more constant and the dynamic response to current changes is faster. Due to the integrated feedback loop, the falsification of the result by the measurement is reduced.

To avoid such measurement feedbacks, Antonopoulos et al. [APS+09] use a measurement setup with a current mirror circuit. There, the measurement is made by integrating a shunt resistor into a mirrored current so that no voltage drop across a shunt resistor occurs in the original circuit. Another more specialised current mirror-based measurement setup is presented by Konstantakos et al. [KKNL06].

3.2 Structures of Energy-Aware Simulations

There are already several network simulators for WSNs. Merrett et al. [MWHA09] examine some of them and identify which criteria are crucial for energy-aware simulations. A fundamental finding of Merrett et al. is the importance of flexibility and adaptability of the simulation. To support the simulation of a variety of different scenarios, the authors suggest that the simulation software must be extensible so that new hardware modules can be easily integrated and specified. The hardware components can be classified in three categories: *Energy Sources*, *Energy Stores* and *Energy Consumers*. *Energy Sources* are for example solar cells, vibration harvesters or mains supply. By *Energy Stores*, Merrett et al. mainly mean different types of batteries, supercapacitors and fuel cells. *Energy Consumers* are all components of the WSN node which consume energy like microcontrollers, radio transceivers, sensors or other peripherals. Besides precise hardware models, Merrett et al. find that an evaluation of the node's software is also important for precise energy modelling. Since different protocols and algorithms require different active times of the node, this can result in large differences in the simulation of the node's lifetime. Compared to existing simulations, Merrett et al. provide environment interaction for the sensor nodes. For example, the environment-aware *Energy-Sources* in the sensor nodes like photovoltaic or wind harvesters are controlled by environment defined models. This also applies to some sensor types like photodiodes. For the energy consumption, all connected modules are considered including any periphery.

Their proposed structure for energy-aware simulations is shown in Figure 3.2. The *Environment Model* contains a *Network Model* that defines the physical location of the nodes, a *Channel Model* that defines the wireless data propagation and an *Environmental Model* that defines relevant physical parameters like sun, wind or temperature. Each node has access to the *Hardware Models* corresponding to its configuration. These *Hardware Models* contain *Energy Models*, *Sensor Models* and *Timing Models* which are able to query the global *Environment Model* as described above. The node's individual *Shared Application Layer* interacts with the *Communication Stack*, the *Energy Stack* and the *Sensing Stack*. These are divided into several sub-layers that represent the whole chain from physical data to processable information and interaction interfaces. For example, the *Energy-Stack* is divided into three sub-layers. First, the *Physical Energy Layer (PHE)* reads the current battery voltage of the *Energy Model*. The *Energy Analysis Layer (EAN)* uses the specific battery and super-capacitor models to convert the voltage data into residual energy and lifetime information. These two layers are called by the *Energy Control Layer (ECO)* which controls the energy-aware operations of the WSN node. The *Communications Stack* and the *Sensing Stack* work basically similar. Using these stacks with the *Shared Application Layer* and the specific models, each node can be individually configured and simulated.

[1] https://arduino-projekte.info/stromverbrauch-arduino-wemos-boards/

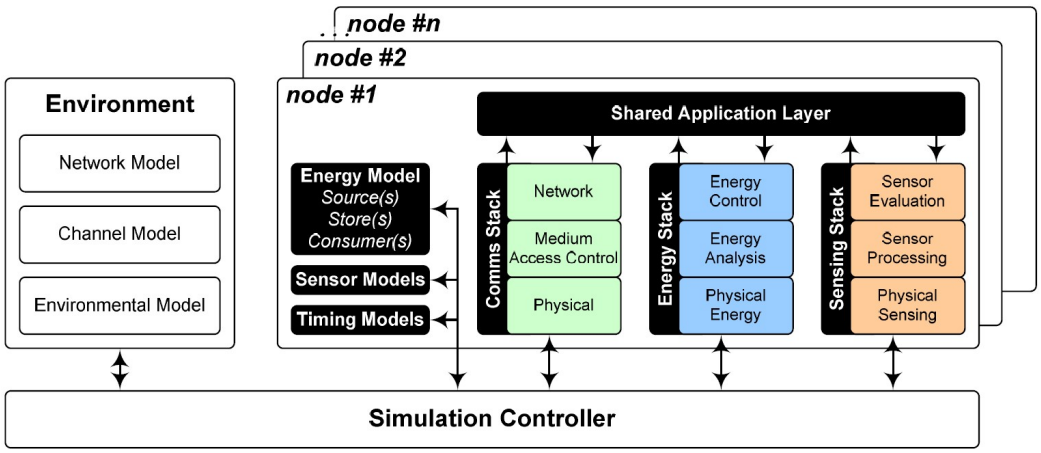


Figure 3.2: Merrett et al. proposed structure for energy-aware WSN simulations [MWHA09].

3.3 Analysis of Related Work

Since this work consists of many different fields, there is already a lot of related work in most of the areas. Especially the area of battery models and energy consumption of WSN-nodes is well explored. As some of the models are very detailed and include many non-ideal effects, simplification or compilation of different models may be sufficient for this work. For example, some battery models cover the Recovery-Effect, but this is sometimes neglected for WSN applications [KPA+10]. Current-voltage behaviour of batteries also seems to be very individual, so that there are only a few works with generally valid data on the voltage drop of batteries. Since a modular system is being developed in this work, a generic model must be developed from the few existing data in terms of voltage drop.

Depending on the required precision, lifetime prediction for WSN-nodes needs in-depth information on the used hardware modules. Due to the modularity aspect in this work, the approaches presented here can only be used to a limited extent. The software or the activities of the node also play a major role in energy consumption, which also requires individual consideration of the corresponding nodes in terms of modularity.

Since Merrett et al. [MWHA09] already put an emphasis on extensibility, they present a useful template for the structure of a good energy-aware simulation. Therefore, this structure can in principle be transferred to this work, even if further adaptations are necessary.

3.4 Summary

In this chapter already existing related work on different areas of this work is presented. Section 3.1 deals with different energy models for batteries and power consumption of typical WSN-node hardware modules. Covering non-idealities of batteries, different battery models, battery lifetime prediction, energy consumption of typical WSN-node components and energy measurement this area is well explored and treated in a lot of other works. Regarding energy-aware simulation structure, there is only one but good adaptable work in section 3.2.

However, most of the related work cannot be adopted directly. In particular, the analysis shows that a precise prediction of the battery life is challenging, especially due to the aspect of modular construction.

4 Design

This chapter describes a new model for modular energy-aware simulations of Wireless Sensor Networks (WSNs). Thereby, the goal of this model is to enable a simulation to consider the energy consumption of the sensor nodes, which are configured as a composition of different components. Due to most of these sensor nodes being typically battery-powered, using this new model, an energy-based lifetime prediction of the sensor nodes can be achieved. To ensure that this model can be used for a variety of different WSN simulation applications, it is designed to be highly flexible and expandable. This is provided by the completely modular structure of the sensor nodes, which are described as a composition of different hardware components. Thus, the model does not describe a single closed systems, but individual components which are assembled into a system as desired. Since sensor nodes in WSNs usually consist of different hardware modules, this modular approach to modelling energy consumption is particularly interesting for WSNs. This also allows hardware modules to be modelled only once but used in different compositions in several individual sensor node configurations. The ability to reuse hardware models in different sensor node configurations makes it possible to quickly, flexibly and easily simulate new hardware compositions in terms of their energy consumption and battery lifetime. This enables fast comparison of different hardware configurations in a simulation environment. Due to the fully modular design, the presented model also offers the integration of new components that may have functions that go beyond the typical tasks performed by sensor nodes in WSNs.

Section 4.1 discusses the requirements on this design and the assumptions made to meet these. Section 4.2 gives a detailed overview over the models system design. For evaluation purpose and creating detailed hardware models, Section 4.3 shows a measurement setup design which meets the requirements of the presented model. Finally, Section 4.4 summarizes this chapter.

4.1 Requirements and Assumptions

As the presented design should be highly flexible and extendible, it requires a modular and configurable basic structure. Sensor nodes have to be freely composable from different hardware components. To ensure the model to be extendible with a variety of different hardware, some simplifications have to be done. For this reason the following assumptions are made:

- All WSN nodes can be configured individually and independently of each other.
- The supply voltage provided by the battery and the current consumption of individual tasks performed by specific modules are considered as constant. This means that the use of model- and task-specific U-I characteristics is not possible.
- Focusing on a modular approach, it is assumed that the set of connected hardware modules increase the total energy consumption of the whole node, but not influence the energy consumption of other connected hardware modules.

This improves the flexibility and strengthens the ability to quickly try out different hardware configurations, knowing that this might decrease the accuracy of the simulation results. Despite these assumptions and simplifications, the model should be able to predict the lifetime of the individual sensor nodes as accurately and precisely as possible. For this purpose, a concrete evaluation of a sensor node lifetime simulation is provided in Chapter 6.

4.2 System Overview

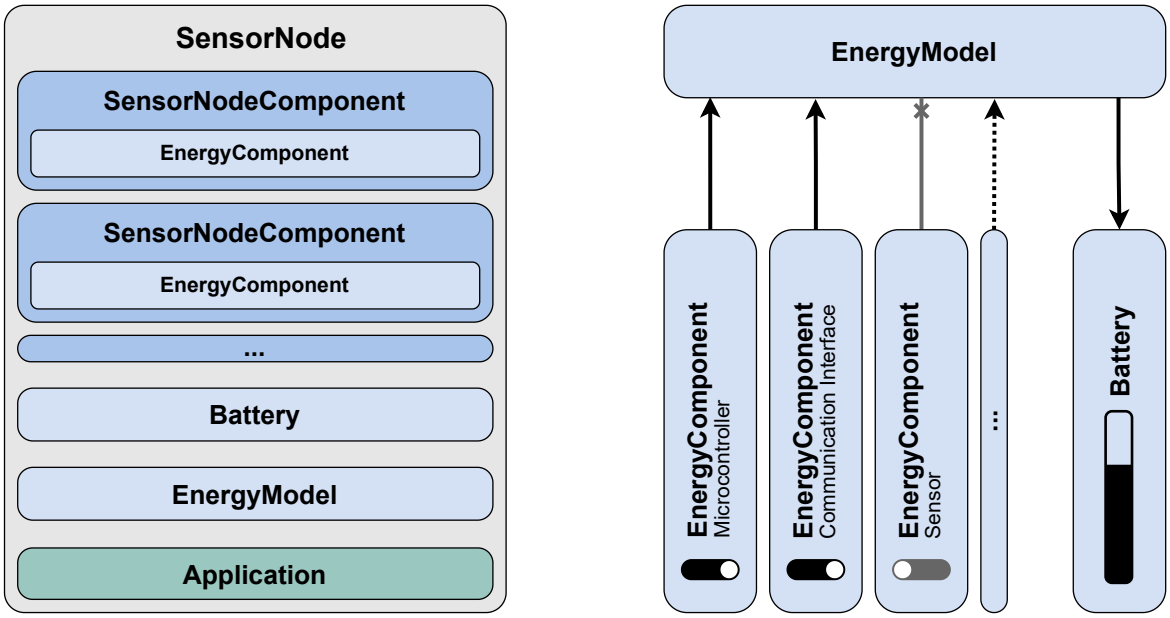
WSNs consist of several sensor nodes. These sensor nodes are self-sufficient in terms of energy and workload. This means, that in a typical WSN scenario, for example, the sensor nodes are able to take

measurements and transmit data on its own, without a central control unit. This enables the model to consider each sensor node separately as an independent unit.

In order for these sensor nodes to operate autonomously, they consist of different components with different functions. As shown in Figure 4.1a, a sensor node contains several *Sensor Node Components*, a *Battery*, an *Energy Model* and an *Application*. *Sensor Node Components* represent hardware modules used in a sensor node like Microcontrollers (MCs), communication interfaces, sensors, actuators and other components. MCs here have a special role, as they are typically the hardware module where every other hardware component is connected to. However, since it would theoretical be possible to equip a sensor node with several MCs, they are treated as normal *Sensor Node Components*. To ensure that the sensor nodes can be freely and individually equipped, any number of *Sensor Node Components* can be added to a sensor node.

Since most sensor nodes have only one battery or one energy source consisting of several battery packs, batteries are not considered as *Sensor Node Components*. This also simplifies the calculation of the current state of charge, as there is a central battery model that represents all connected hardware batteries of the sensor node.

The *Application* component manages the workload inside a sensor node. This means the *Application* activates, configures and controls the individual *Sensor Node Components*. Most *Sensor Node Components* represent actual hardware modules that consume energy. Therefore, they bring matching energy components with them. These *Energy Components* define the actual energy consumption of all the tasks which can be performed by their corresponding *Sensor Node*. Together with the *Energy Model* the sensor nodes current total consumption is then calculated from the individual consumptions of the *Host Components*, which is then taken from the battery.



(a) Basic structure of a sensor node. A sensor node contains several *Sensor Node Components*, a *Battery*, an *Energy Model* and an *Application*.

(b) The *Energy Model* handles the energy consumption process for all connected *Energy Components*. Components are able to be switched on and off.

Figure 4.1: Basic structure of a sensor node and its energy management.

Figure 4.1b shows the basic concept of energy consumption within a *SensorNode*. The individual *Energy Components* of the corresponding *Sensor Node Components* notify the *Energy Model* if they consumed energy. Defining the energy consumption of different tasks and features of the *Sensor Node Components*, *Energy Components* have several states. By changing state, the *Sensor Node Components* adjust their

corresponding *Energy Component* to fit the energy consumption to the current task. The tasks of the *Sensor Node Components* are controlled by the *Application*.

The energy consumptions of all *Sensor Node Components* is directed to the *Energy Model*. Based on the incoming consumptions, the total current energy consumption of the sensor node is calculated. The *Energy Model* finally discharges the *Battery* with the calculated amount of energy.

4.2.1 Sensor Node and Energy Components

The *Sensor Nodes* stand as containers for single individual sensor node configurations inside a WSN, including a *Battery*, an *Energy Model*, an *Application* and several *Sensor Node Components*. These *Sensor Node Components* represent all modules that are physically connected to the *Host*. In most scenarios, the central hardware component of a sensor node is an MC. Other typical hardware components of sensor nodes are sensors, communication modules, motors, displays, LEDs or other periphery components. In a real hardware sensor node, all other components are typically managed by an MC. In principle it is also possible to add several MCs to a sensor node, although this is rather unusual, as MC are often among the most energy-intensive components of a sensor node. This modular design supports several MCs connected to a sensor node. However, different applications running on these MCs have to be merged into one application module.

Depending on the complexity of a specific hardware module, a component is described as a Finite State Machine (FSM) with at least two or more states. Components can be switched on and off, or consist of more states to be described accurately in terms of their logical behaviour.

Due to most hardware components consuming energy while being turned on, *Sensor Node Components* require a corresponding *Energy Component*. These *Energy Components* define the energy consumption the *Sensor Node Component* requires for its task in its current state. As mentioned in Section 3.1.4, several papers examine single hardware component modules to find out the corresponding energy consumption of the module in a specific state. These specific states often differ greatly in their energy consumption, as many hardware modules support several active and sleep states with different features and functionality. Some MCs even have several different sleep states with different grades of power consumption and functionality. In this design, the states of the sensor nodes are defined at task level. This means that typical WSN tasks such as sampling sensor data or transmitting data are treated as individual states. Some related works examine these states in more detail and subdivide the states used here into more detailed sub-states. Since these sub-states must all be passed through to complete this state on task level, a more detailed model brings no more information about energy consumption since the amount of consumed energy does not change. From the point of view of energy consumption simulation, it is therefore equivalent to define the energy consumption of a single state with an average energy amount or a power consumption and the corresponding duration spend in this state.

Figure 4.2 shows the basic workflow of the energy management of a sensor node. Starting with the *Application* that regularly checks the battery's state of charge, it successively performs various tasks from the configured workload. For this purpose the *Application* sets the *Sensor Node Components* to their corresponding state. The *Sensor Node Components* again set their corresponding *Energy Components* to consume the corresponding energy. These energy consumptions are reported to the *Energy Model*, that discharges the *Battery* with the total energy consumption.

4.2.2 Energy Model

The *Energy Model* coordinates the nodes' energy consumption. By adding all consumption values of the active components, the *Energy Model* calculates the total current energy consumption of the node. Due to different states in the applications workload cycle, the energy consumption of a node is not constant. Due to the chosen energy-based approach, the correct current total energy consumption is calculated by

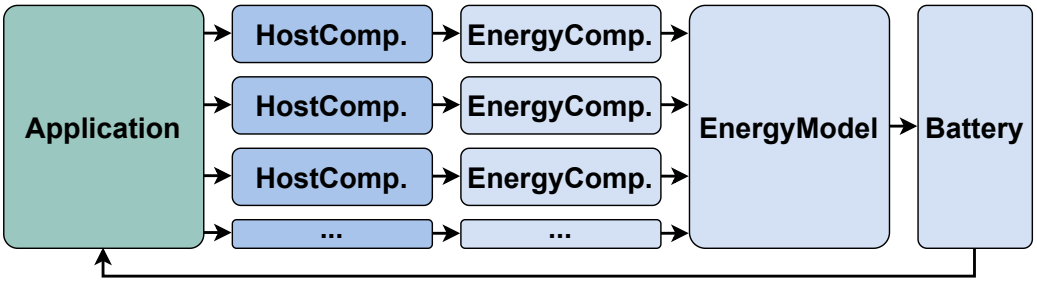


Figure 4.2: Procedure of energy consumption in the sensor node model.

the *Energy Model* by adding up the energy consumptions of all *Sensor Node Components* defined by their corresponding *Energy Components*.

Another approach would be to examine the current consumption of the components and discharge the battery with the sum of these currents. Since different hardware components need different supply voltages, information about the current only are not sufficient in terms of energy consumption. Adding information about the corresponding voltage, the model would effectively get information about power consumptions. Adding a time component is also necessary to ensure the correct duration of the consumption. Finally, the transferred information at the battery would again result in an energy information. Using information about energy simplifies the *Energy Model* and also supports the usage of simple *Battery Models* with a given extractable energy in *Joule*.

Since energy is an absolute and time-independent value, energy consumption is defined for a specific duration in a certain state. It represents the whole energy that is needed to perform the corresponding task. This full amount of energy can be taken from the battery at different times. One possibility is to deplete the battery, when the state changes. This can be done, when the current state starts or when it ends. Since this model should rather model a too short than a too long sensor node lifetime, it is recommended at this point to discharge the associated energy already at the beginning of a new state. Due to the times at which the state changes are usually not evenly but unevenly distributed, this strategy is called *Dynamic Pushing*. This strategy is well-suited for applications with many and steady state changes, as the consumed energy is taken from the battery regularly. However, this state change-based energy consumption calculation is unsuitable for applications with long periods without state changes. Discharging the energy at the beginning of a long new state can lead to inaccuracies because the simulation stops when the energy of the entire new state is discharged at its beginning. This means that the simulated lifetime would be too short, or too long if the energy is used up at the end of a state.

Another approach fixing this issue would be an energy depletion strategy with a specific static time interval at which energy is taken from the battery. Due to the static time interval, this approach is called *Static Pushing*. The shorter the interval, the higher is the respective accuracy of the model. If the interval is too long, rapid changes of state may not be calculated correctly because other states than the current one were also active in the past time interval. On the other hand, a short interval might lead to a weaker simulation performance since these calculation activities must be done for every sensor node of the WSN independently. Further, the energy consumption of the states must be split with regard to the query interval, to ensure the correct amount of energy is taken from the battery.

To overcome both issues of inaccurate energy depletion, a hybrid approach combining static and dynamic behaviour is proposed. As shown in Figure 4.3, energy is taken from the battery when the state changes and when the static query interval is over. This enables the model to use the advantages of both strategies. The static query intervals can be longer without missing rapid changes of state. This can reduce the computing effort even for large networks.

For applications with continuous and frequent state changes, the static query interval is not as important as for applications with infrequent state changes. Due to state changes triggering the energy model,

applications with frequent state changes take energy from the battery regularly for the time interval in the last state.

There are several further ideas to optimize this strategy which may especially interesting for applications with a high amount of modelled sensor nodes. For example, it would be conceivable to change this static query interval dynamically depending on the current state of charge of the battery. Short intervals only increase the precision at the end of the simulation where the battery might get fully discharged by the next energy consumption query. If the battery is still full, shorter query intervals only increase the computational effort. Depending on the number of simulated nodes and the requested precision, this query interval can be shorter or longer.

Overall, to use *Static-* or *Hybrid Pushing*, it is necessary to provide an average energy consumption per time for the individual states, which is equivalent to an average power consumption.

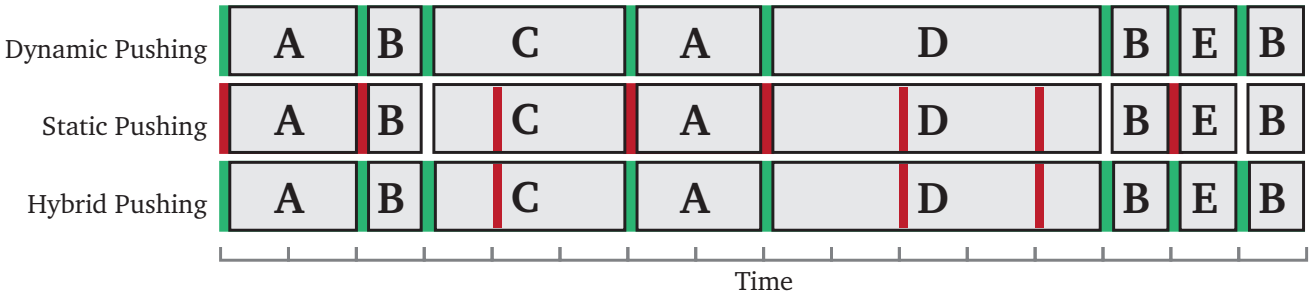


Figure 4.3: Visualisation of *Dynamic*, *Static* and *Hybrid Pushing*. *Dynamic Pushing* discharges the battery before each state change. *Static Pushing* discharges the battery in a fixed time interval. *Hybrid Pushing* combines both methods. Here, the *Static Pushing* interval is set to two time steps. The state sequence is exemplary.

4.2.3 Battery

As discussed in Section 3.1, batteries are central components of sensor nodes and decisive with regard to the sensor nodes' lifetime. Due to the modular design of the sensor node model, different battery models can be used to describe the sensor nodes energy source. For example, the precise battery models discussed in Section 3.1.2 such as the Kinetic Battery Model (KiBaM) or the Stochastic Battery Model (StBaM) are ideal for well-studied batteries or batteries with detailed data sheets. Since several parameters are required to specify individual battery behaviour, these battery models require great knowledge about the corresponding battery behaviour. As deep knowledge and battery-specific characteristics may not be available for all batteries, a battery model as generic as possible is best-suited as a default battery model in this sensor node model. That means that the user can configure the total usable energy provided by the used battery and gets a corresponding battery model for the WSN simulation. Effects like *Peukert's Law* or voltage drop can be considered in a generic model as well. However, using an energy-based approach, the current voltage of the battery has no influence on the energy management. If the specified amount of usable energy is used up, the battery is considered fully discharged. Due to the simplified configuration, this simple battery model provides a estimate about the batteries behaviour. However, the precision of the simulated sensor nodes lifetime using a simple generic battery model will not be as accurate as with the precise battery-specific models.

Since batteries are typically labelled with an electrical charge rating in mA h or an energy rating in W h these information need to be converted in a corresponding energy in *Joule*.

Assuming constant voltage and current, Listing 4.1 can be used to transform power and time to energy in *Joule*.

$$E = \int_{t_0}^{t_1} P(t)\,dt = P_{const} \cdot \Delta T \text{ with } \Delta T = t_1 - t_0 \tag{4.1}$$

With this assumption made and $P_{const} = U_{const} \cdot I_{const}$, Listing 4.2 and 4.3 show the conversion from electrical charge Q to electrical energy E.

$$Q = \int_{t_0}^{t_1} I(t)\,dt = I_{const} \cdot \Delta T \text{ with } \Delta T = t_1 - t_0 \tag{4.2}$$

$$E = U_{const} \cdot I_{const} \cdot \Delta T = Q \cdot U_{const} \tag{4.3}$$

The values from the battery in mA h multiplied with a voltage V can be converted in SI units with $1\,\mathrm{mA\,h\,V} = 3600\,\mathrm{mA\,s\,V} = 3.6\,\mathrm{A\,s\,V} = 3.6\,\mathrm{W\,s} = 3.6\,\mathrm{J}$. W h is already a unit for energy and can be converted in *Joule* by using $1\,\mathrm{W\,h} = 3600\,\mathrm{W\,s} = 3600\,\mathrm{J}$.

More precise battery models can be implemented by adding non-idealities to the battery model like *Peukert's Law* or the *Recovery-Effect*, as mentioned in Section 3.1.1. This can be done by using typical battery models from Section 3.1.2. Also the battery voltage can be examined in detail by adding a voltage drop behaviour in the battery model as described in Section 3.1.3. Therefore, a given or measured *U-I*-characteristic has to be added to the model. The use of these extended models would improve the simulated battery behaviour, which increases the accuracy of the simulated battery lifetime. On the other hand, these models require in-depth knowledge of the batteries used, which can only be achieved through detailed data sheets or elaborate measurements.

4.2.4 Application

The *Application* of a *Host* manages the tasks that are performed by the node and the individual corresponding *Sensor Node Components*. For this reason, the *Application* repeats an *Application Cycle* that includes all tasks that should be performed by the *Host*. Figure 4.4 shows a typical WSN *Application Cycle* example with the tasks wake-up, sampling sensor data, processing these data, sending data via wireless transmission and going to sleep after. This *Application Cycle* is repeated until the battery is fully discharged. The individual tasks in the cycle differ in duration and active *Sensor Node Components*. This results in different amounts of energy consumption in these states. To ensure the states of the individual *Sensor Node Components* fit the current *Application Cycle* state, the *Application* is able to set the states of all *Sensor Node Components*. Basically, each application state contains control information for all connected *Sensor Node Components*.

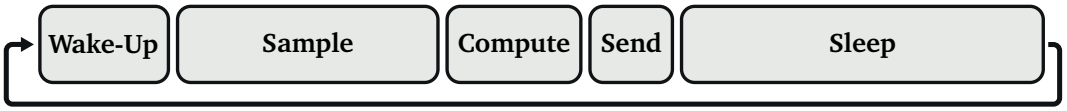


Figure 4.4: An example for an *Application Cycle* with tasks of different duration. This cycle is repeated until the *Battery* of the *Host* is completely discharged.

As the proposed model has the goal of high flexibility and modularity, the application of each sensor node is configurable independently. Also, the model is extendible with other user-defined applications and application tasks.

4.3 Acquiring Real-World Data for parameterised Simulation Models

The model describes in the previous sections is based on a modular structure. This allows the user to combine and evaluate the composition of different hardware modules within a sensor node. For a meaningful evaluation of the energy configurations of sensor nodes in a simulation, the individual hardware modules used in the proposed WSN model need to be parametrised to resemble their real-world counterparts. Also, for the evaluation of this design, it is necessary to compare the simulated sensor node lifetime with the lifetime of a corresponding real-world sensor node.

Due to the modular approach, instead of measuring the whole sensor node with different applications, each component used in the evaluated simulation is isolated and measured for itself. In cases of sensors or other isolated hardware components, voltage and current can be measured on the modules itself. With MCs and components with several functionalities built in, an isolated measurement is not always possible. Instead, the current consumption can be evaluated with a programmable MC that turns the module to be measured on and off. The difference of these two current values can be considered as the current consumption of the module. Since the supply voltage of these modules stay constant, a temporal measurement of the voltage is not needed. As shown in Listing 4.4, the power consumption P can be calculated by integrating the voltage and current over time and dividing the result by the time interval $\Delta T = t_1 - t_0$. Assuming constant voltage $U(t) = U_{const}$ and current $I(t) = I_{const}$, the power consumption formula can be reduced to $P = U_{const} \cdot I_{const}$.

$$P = \frac{1}{\Delta T} \int_{t_0}^{t_1} u(t) \cdot i(t)\, dt = U_{const} \cdot I_{const} \tag{4.4}$$

As described in Section 3.1.5, there are different methods of measuring current and voltage of hardware components. By using a shunt resistor in series with the load, as shown in Figure 4.5a, it is possible to measure the voltage U_{sh} and calculate the corresponding current I_L. Assuming a shunt resistor with resistance under 1 Ω, the current flowing through the voltmeter can be neglected, as these typically have an input resistance greater than 1 MΩ. Based on *Ohm's Law*, the voltage drop over the shunt resistor U_{sh} can be determined by adjusting the resistance of the shunt resistor R_{sh} to the expected current I_L, as shown in Listing 4.5. Using the *four-wire sensing* method for measuring small resistors[1], the shunt resistor can be determined precisely, which is important as the shunt resistance influences the calculated current with $1/R_{sh}$.

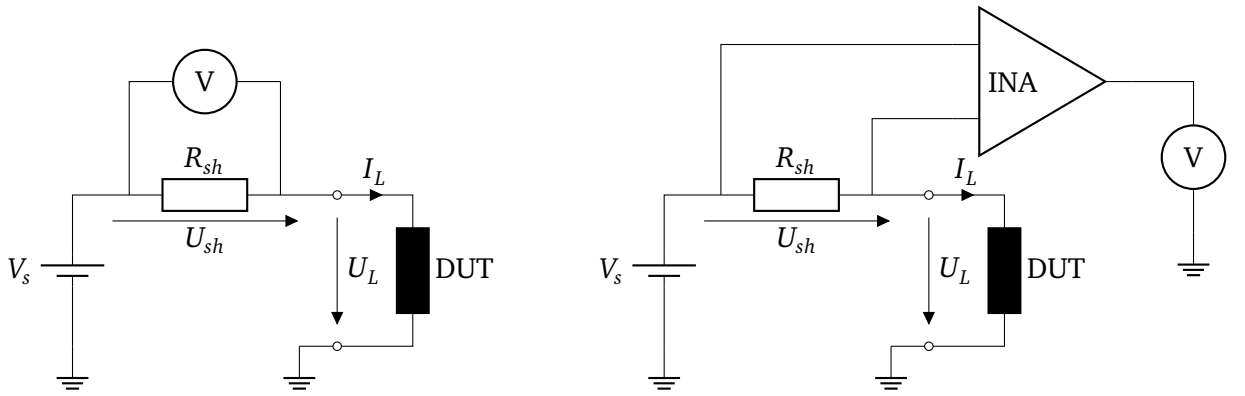
$$U_{sh} = R_{sh} \cdot I_L \Leftrightarrow I_L = \frac{U_{sh}}{R_{sh}} \tag{4.5}$$

Since the shunt resistor in series is equivalent to a voltage divider, the voltage drop U_{sh} should not be too big, as it decreases the voltage over the measurement object U_L. If the voltage U_L falls below a certain threshold, the Device Under Test (DUT) may not work properly anymore. In addition, the power consumption of the module may change if the supply voltage deviates from the nominal voltage. On the other hand, using the same measurement device, the precision of the measured shunt voltage increases with a greater amount of shunt voltage. To improve the measurement, an Instrumentation Amplifier (INA) can be added to the measurement circuit as shown in Figure 4.5b. INAs are electrical amplification circuits that are especially known for their high impedance input resistors. As a result, a negligibly small current flows through the parallel measuring circuit, so that the measurement result is not falsified. Amplifying the input differential voltage, the INA outputs the same voltage as measured in Figure 4.5a but with a higher gain. By using an INA, a smaller shunt resistor can be used by achieving at least the same amount of measurable voltage over the voltmeter. This enables the DUT to have a greater range of measurable current consumption without significantly decreasing the supply voltage V_L. This approach is easy to implement using commercial available measurement boards like the *INA219* by *Texas Instruments*[2].

4.4 Summary

In this chapter, the basic design of the modular energy-aware simulation model is presented. Generally, due to the request of flexibility, modularity and extensibility, the model partially uses simplified component models. Also, generic models for different components are presented, which makes the simulation

[1] https://xdevs.com/doc/Keithley/Appnotes/2Wire_4Wire%20Resistance%20Article.pdf
[2] https://www.ti.com/lit/ds/symlink/ina219.pdf



(a) Measurement setup with a shunt resistor. **(b)** Measurement setup with shunt resistor and INA.

Figure 4.5: Measurement setups with shunt resistors. DUT represents the respective hardware module under test. V_S is the nominal supply voltage for the DUT.

easy to configure and provides fast results without deep knowledge of the used hardware components, especially the batteries. This generic approach in its default configuration is not best-suited for high precision simulation. However, due to the modularity and extensibility, hardware specific battery or component models can be implemented easily so that the simulation is able to provide results with higher precision as well.

The main structure of the simulation model contains *Hosts* for individual WSN nodes. These *Hosts* are containers for several *Sensor Node Components*, a *Battery*, an *Energy Model* and an *Application*. The flexible use of a generic, predefined and extensible custom *Sensor Node Components* enables this design to support a modular *Sensor Node* architecture. A downside of this approach is that for precise simulation the model needs further adjustments, especially when using the heavily simplified generic battery model. Additionally, the proposed model can easily be extended with specific and further developed components.

Finally, a measurement setup is presented, that is suitable for measuring real-world data of typical sensor node hardware. By using this setup with an INA amplifying the shunt voltage, the measurement capabilities are improved. These measurements are necessary to define the performance characteristics of new hardware components and to parameterise their respective models for simulative evaluation.

After discussing the concepts of the energy-aware model for sensor node simulation in this chapter, details of the implementation of this model are presented in the following.

5 Implementation

After the last chapter presented the proposed design from a conceptual view, a prototypical implementation of the proposed model is presented in this chapter. The goal is to demonstrate the feasibility and give an example of how this proposed design can be implemented. For this, the presented model is used to extend a simulation framework for wireless networks to enable modular energy-aware sensor node lifetime simulations.

The used simulation framework *Simonstrator*[RSRS15] provides different functionalities in terms of energy consumption and module interaction, which are extended and adapted based on our conceptual design. These adjustments and further general design decisions are discussed in Section 5.1. Afterwards, Section 5.2 provides detailed insights into the architecture of this implementation. This also includes an overview of the most important and central interfaces and classes. Section 5.3 explains how the interaction of the individual parts and components is implemented in this work. Before the summary of this chapter in Section 5.5, Section 5.4 analyses the limitations of this concrete implementation.

5.1 Design Decisions

To be able to use the presented model for the simulation of sensor nodes and their lifetime, it is determined how the proposed design from Chapter 4 can be implemented best. For this purpose, several decisions and considerations are made about the concrete implementation of the model. Due to external conditions, such as those imposed by the used simulation framework, various changes have to be made to the model for implementation. This section explains different decisions regarding the used simulation framework and the resulting opportunities and constraints. In addition, general implementation-specific design decisions and the parameter selection for generic components are discussed.

5.1.1 The *Simonstrator* Simulation Environment

To be able to use the model presented here for the simulation of sensor nodes, a simulation framework is required in which the model can be embedded. In general, various simulators can be used for this purpose. Since the Java-based network simulation framework *Simonstrator* [RSRS15] is particularly suitable for distributed systems that consist of several different components, it is used in this implementation as the simulation environment for the proposed model. Due to the component-based approach, the *Simonstrator* provides access to the components and modelled hardware modules of individual simulated network devices. This — and the support for mobile networking — enables the individual and independent simulation of multiple sensor nodes in a Wireless Sensor Network (WSN). The *Simonstrator* has a wide range of basic functions, such as pre-built transmission models that can be used in the implementation of the proposed design. Further, the *Simonstrator* already comes with a basic energy simulation approach. As shown in Figure 5.1, the basis of this existing approach is an *Energy Model*, where multiple *Energy Components* can register. These different *Energy Components* again contain a certain number of *Energy States*. Each *Energy State* defines a specific energy consumption, with which the *Energy Component* can calculate its current energy consumption. After getting notified by an *Energy Component* consuming energy, the *Energy Model* notifies the battery to reduce the amount of preconfigured available charging capacity. The implemented *Simple Battery* model does not consider any non-idealities or other advanced concepts of battery modelling. This basic concept can be excellently extended by the design presented.

Independent of the existing energy management approach, the *Simonstrator* framework provides event-based notification methods to interact between different components of the simulation model.

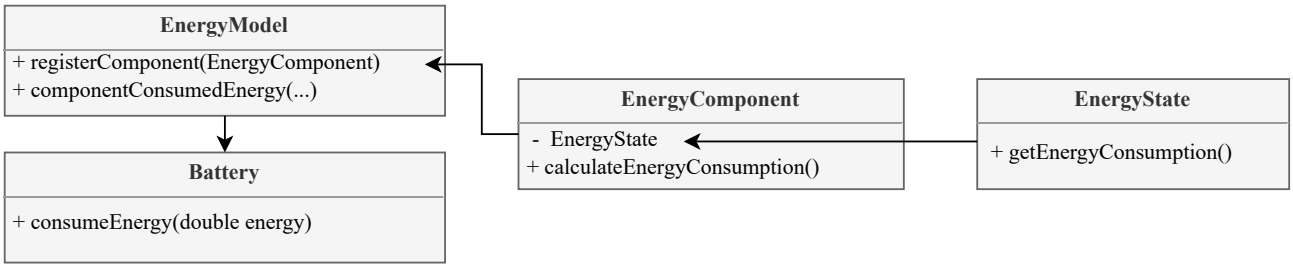


Figure 5.1: Simplified structure of the existing approach for energy simulation in the *Simonstrator* framework.

Thereby, the framework supports events to be performed immediately and events to be performed with a certain delay. More information about how the different components of the design interact with each other are given in Section 5.3.

To configure a simulation in the *Simonstrator*, an *XML* file structure is used to define the network and the included sensor nodes. In addition to the pure network properties, various parameters can be set, for example the simulation properties, the topology and node positioning, the behaviour of mobile nodes or visualisation and evaluation settings. Thereby, sensor nodes are organized in groups. The sensor node properties and energy modules like the *Energy Model* are configured for each group of sensor nodes individually. This means, sensor nodes in the same group have the same configuration. Since there is no limit to the number of groups, the number of sensor nodes with different configurations is also not limited. After starting the simulation, corresponding Java objects are created and initialized from the parameters defined in the *XML* file structure.

Due to the use of the *Simonstrator* as a suitable simulation environment, many models and concepts from the proposed design can be implemented directly. However, in addition to the advantages of this simulator, there are also a few limitations. The handling with these and further general implementation-specific design decisions are explained in the next section.

5.1.2 Implementation-Specific Design Decisions

Since the model design allows different possible implementations, decisions have to be made about the concrete model implementation. This means, on the one hand, adapting the model to enable interaction with the simulation framework. On the other hand, general decisions about the concrete implementation have to be made, due to the design allows a variety of different implementations. Further, some concepts of the design are not realisable due to different limitations and have to be implemented in another way. Some examples of the most important design decisions made for this implementation are explained in the following:

- As described in Section 5.1.1, the *Simonstrator* parses the whole simulation configuration from an *XML* structure into Java objects. Therefore, corresponding Java classes and parameters for the construction of the objects have to be defined. To reduce the number of parameters to be set in the *XML* structure, class-specific *Factory* classes are used. Reduced to the handover of different component-specific characteristic parameters, the *Factory* classes can be used to manage the provision of any object parameters required for their corresponding object construction. This enables the configuration effort to be reduced to the crucial parameters, which makes the configuration faster and the *XML* structure clearer.
- Since different objects need to interact with each other, often after a certain event, an event-based notification interface is used. This means that objects that can be triggered externally by other objects, implement an event handler interface that is used to define the further procedure within the triggered object. The advantage of using such events is that they can also be triggered with a

user-defined delay. Since the *Simonstrator* framework provides a time interface, these event delays are handled by the simulation framework to ensure correct synchronized timing. Details on the interaction of components and event-based object interaction are given in Section 5.3.

- To enable the simulation to return the duration from a host starting for the first time until it is deactivated due to a completely discharged battery, a timer needs to be implemented. Since this timer functionality is required for all sensor nodes, there are several options for concrete implementation. One would be to implement a global sensor node timing module that measures the duration of all sensor nodes. This would enable the timer at the end of the simulation to easily give a centralized view of all sensor node lifetimes. However, this means that all sensor nodes have to register at the global timer module and send start and end events together with a unique id to enable the timer module to measure the individual durations accurately. In this implementation, another approach is used where the individual applications of the sensor nodes themselves implement this timer functionality. This decentralized approach gives the applications of the sensor nodes direct access to their current lifetime. This enables an easy, decentralized and accurate sensor-node specific time evaluation.
- When implementing sensor components, the question arises as to which data should be used as sampled sensor data. Since the *Simonstrator* does not bring any kind of environmental model, the sensor components are not able to get meaningful values from the simulation. However, for the simulation of the sensor node's battery lifetime, meaningful sensor data are not required. Since only the amount of bytes transmitted is relevant for the energy consumption of the transmission model, the sensor components generate random data that correspond to the expected transmission size of the associated hardware sensor modules.

Despite a fairly design-oriented implementation, some concepts are not compatible with the framework used or have to be implemented differently than originally designed for other reasons. Design decisions that address these limitations are discussed in the following:

- Different than defined in the presented design, the battery component in the *Simonstrators* basic energy concept is configured as a part of the energy model. To support the modular aspect, the presented design provides that all components of the sensor node model are defined as a list of components with user-defined length. However, since the design also envisages that only one battery or battery model is used per sensor node, it makes no functional difference to the implementation if the *Battery* is configured as part of the *Energy Model* or as a normal *Host Component*, even if this means that it might not support the standardised methods of the *Host Component* interface.
  The same applies to the configuration of the transmission technology. As shown in Listing 5.1, the *HeltecESP32LoRaConfig* configuration is defined as a component inside the *EnergyModel* tag. This is necessary, as the used predefined *EnergyModelFactory* of the *Simonstrator* framework requires such a configuration in order to ensure the interplay of the transmission component with the network model. All other components, such as the sensor node's *Applications*, *Sensors*, *Microcontrollers (MCs)* and other peripherals are defined by adding the corresponding components to a list inside a specific *Hostbuilder* section in the *XML* structure.
- The consumptions of the individual *Host Components* inside the sensor nodes, are specified as a power consumption in microwatt, stored in the corresponding *Energy States*. The energy consumption in microjoule is firstly calculated in the *Energy Component* by multiplying the power with the duration of the consumption. This decision is based on the proposed energy-based consumption approach is limited to fixed state durations. Since the *Application* and the used *Energy Components* are configured independently, for a general simulation configuration it is more flexible to define the energy consumption of the components independent of their individual state durations. This results in specifying the energy consumption as an energy consumption per second, which is equal to a power consumption.

```
<Default>
   <Variable name="BATTERY_CAPACITY_MAH"    value="4500"/> <!-- mAh -->
   <Variable name="BATTERY_DEFAULT_VOLTAGE"  value="3700"/> <!-- mV  -->
   <Variable name="BATTERY_INITIAL_PERCENTAGE" value="1" /> <!-- decimal 100% = 1 -->
   <Variable name="BATTERY_INITIAL_VOLTAGE"  value="4200"/> <!-- mV  -->
</Default>

<EnergyModel class="de.tud.kom.p2psim.impl.energy.EnergyModelFactory"
   useRandomBatteryStartConfiguration="false">

   <Battery class="de.tud.kom.p2psim.impl.energy.ExtendedBattery"
      cap_mAh="$BATTERY_CAPACITY_MAH"
      defaultVoltage="$BATTERY_DEFAULT_VOLTAGE"
      initialPercentage="$BATTERY_INITIAL_PERCENTAGE"
      initialVoltage="$BATTERY_INITIAL_VOLTAGE"
   />

   <Component class="de.tud.kom.p2psim.impl.energy.configs.HeltecESP32LoRaConfig"
      SF="7"
      transmissionPowerDBm="7"
   />

</EnergyModel>
```

Listing 5.1: Extract from the XML configuration file, showing the configuration of the energy model including the battery and the LoRa configuration of the Heltec Esp32 LoRa board. Variables are used to make the configuration more clearly.

The design decisions discussed here serve as the basis for the rest of the implementation, whose architecture is described in the next section. Among other things, the consequences of certain design decisions are shown in the following section. In addition, Section 5.2 also goes into more detail on some particularly interesting aspects of the implementation.

5.2 Architecture

This section explains how the structure of the presented conceptual design is mapped to the concrete implementation in the Java-based simulation framework *Simonstrator* [RSRS15]. A special focus is placed on the basic architecture, but also on particularly interesting parts of the implementation. Since network models and possibilities for interaction between different *Hosts* are completely implemented in the *Simonstrator*, this section shows the implementation of the inner-Host architecture.

The proposed model from Chapter 4 demands a flexible and modular structure of the sensor nodes. Figure 4.1a from the design chapter shows the desired basic structure and division of the sensor node in different *Host Components*, a *Battery*, an *Energy Model* and an *Application*. As already explained in Section 5.1, this basic structure is adopted directly for most parts. However, due to the reasons and restrictions of the framework explained in Section 5.1.2, some modifications have to be done. This mainly concerns the implementation of the *Battery* and a transmission model configuration as a part of the *Energy Model*. This is necessary as the design decisions are to adapt the model design rather than change the existing *Simonstrator* framework to ensure the continued usability of this work for other projects. Sections 5.2.1 and 5.2.2 show how these further developments are made to implement this model in the existing infrastructure of the *Simonstrator*.

5.2.1 Interfaces

Figure 5.2 shows an extract of the main structure of already existing interfaces and interfaces that are added in this work to the *Simonstrator* framework. When looking at these main interfaces, it is noticeable that the basic elements of the conceptual design can be found almost one-to-one in the structural implementation at the interface level. The fact that some interfaces are subtypes of the *HostComponent* interface and extend it accordingly already shows the implementation of a modular design. Thus, classes that implement one of the subtypes of the *HostComponent* interface, are automatically recognized as an implementation of a *Host Component*. The use of such interfaces can therefore guarantee that certain implementations of concrete components also implement the corresponding methods that are required for the function of this type of component. Further, these interfaces enable the *Host* to classify the components from the configuration. With this, the *Host* can distinguish between the different *HostComponent* subtypes. For this reason, it is important that all classes implement their corresponding interface.

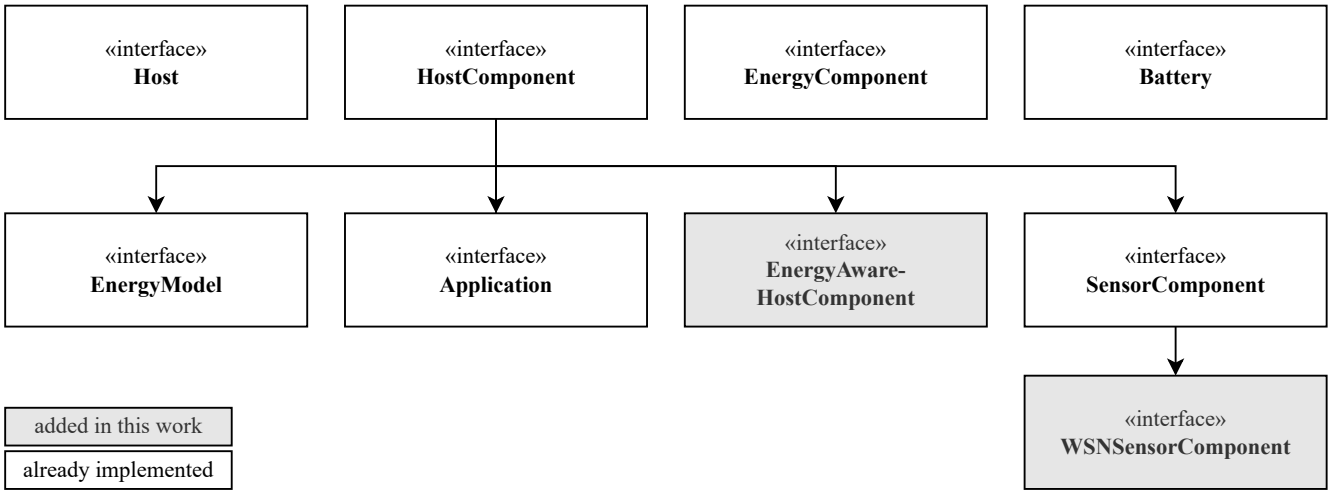


Figure 5.2: Overview over the basic interfaces. *EnergyModel*, *EnergyAwareHostComponent*, *Application* and *WSNSensorComponent* are subtypes of the *HostComponent* interface.

Since most of these interfaces are already implemented in the *Simonstrator*, the focus is now on the two added central interfaces of this work, which are the interfaces *EnergyAwareHostComponent* and *WSNSensorComponent*. The interface *EnergyAwareHostComponent* is a subtype of the *HostComponent* interface and represents *Host Components*, that have an energy consumption greater than zero. This enables for example the *Applications*, to distinguish between *Host Components* that implement just logical behaviour without actual energy consumption and *Host Components* that represent real hardware modules that actually consume energy. This interface is therefore used more as a flag than to define concrete method headers. However, the *EnergyAwareHostComponent* interface defines a setter and getter method for a corresponding *Energy Component*. More details on the usage of this interface inside a concrete implementation of an application are given in Section 5.2.2.

The second central interface added in this work is the *WSNSensorComponent* interface which basically represents a sensor component. Since the *Simonstrator* already comes with a sensor interface called *SensorComponent*, this new interface is a subtype of this existing sensor interface, that also implements the *HostComponent* interface. Similar to the *EnergyAwareHostComponent* interface, this new interface is also mainly used as a flag. Since the *Simonstrator* comes with some types of virtual sensors, this is necessary to differentiate between real WSN hardware sensors and virtual simulation-internal sensors. From the *SensorComponent* interface, the *WSNSensorComponent* inherits the methods `public boolean testSensor()`, `public double getSensorData()`, `public int getSensorDataSize()` and `public String getName()`. The method `testSensor()` is a

method to test whether the sensor is working. The other methods are getter methods that request certain data from the sensor component.

All interfaces that extend the *HostComponent* interface also inherit the following methods: `public void initialize()`, `public void shutdown()` and `public Host getHost()`. The method `initialize()` is called by the corresponding *Host* after all components of the *Host* have been registered. This call is mostly used to prepare the component for the upcoming simulation and registration at other *Host Components*. If a component is de-registered by a *Host*, the `shutdown()` method is called. As further explained in Section 5.3, the method `getHost()` is essential for the interaction between different *Host Components*, as the *Host* is the central element of a sensor node.

Besides these main interfaces, further interfaces have been added for different reasons. For example, the interface *Deactivatable* enables components to be switched on and off. However, these have a low relevant influence on the main architecture of the implementation presented here.

Since interfaces typically do not implement concrete methods, classes are needed to use these interfaces to implement the corresponding functionality. The important classes for the architecture of this implementation are explained in the following section.

5.2.2 Classes

As shown in the previous section, the main architecture of this implementation is close to the proposed design from Chapter 4. Using the presented interfaces, different classes are used to implement the required functionality of the respective components. In particular, the concept of inheritance and the implementation of generic models are used to provide a very high degree of flexibility.

The *Simonstrator* already comes with classes implementing already existing interfaces. While some of these classes can be used for this work, other components require new classes to be written. Basically, the proposed design requires at least one concrete implementation for each of the interfaces shown in Figure 5.2. The already existing class *DefaultHost* is used as an implementation of the *Host* interface. This class manages the different types of *Host Components* by providing several getter, setter and registration methods. This means that after correct configuration and initialisation, the host has different lists with different subtypes of the *HostComponent* interface. This is necessary so that other components, for example the *Application*, can access different components of the same *Host*. The class *WSNSensorApplication* is a new implementation for such an *Application*.

WSNSensorApplication

The class *WSNSensorApplication* is a central component of the implemented design. It provides a configurable implementation of a typical WSN sensor node *Application* workload. This workload includes three basic tasks: sleeping, sampling and transmission. An example *XML* configuration of this *application* is shown in Listing 5.2.

```
<!-- Generic Application component. All time values in micro seconds
<Application class="(...).WSNSensorApplication$Factory"
   activeTimeUs="600000000"
   sleepTimeUs="300000000"
   transmissionIntervalUs="12400000"
   sampleFrequencyUs="250000"
/>
```

Listing 5.2: Configuration of the configurable sensor node application *WSNSensorApplication*

With the first parameters of this application component, *activeTimeUs* and *sleepTimeUs*, the durations of the corresponding states are defined in micro seconds. While all components are deactivated in the sleep state and only the MC consumes energy in its sleep configuration, the active state can be further configured. For this purpose, the next two parameters, *transmissionIntervalUs* and *sampleFrequencyUs*, are used. The parameter *transmissionIntervalUs* defines how often during the active time of the sensor node, a data transmission takes place. In the case of *SampleFrequencyUs* the value in micro seconds defines how often the sensors are used to sample data. Just like the *TransmissionIntervalUs* parameter, this specification is only relevant for the active time of the sensor node. This type of configurable application can be used for basic application consisting of these tasks but different durations.

The concrete implementation of this *Application* consist of a constructor setting the required parameters, getter and setter methods, as well as many methods implementing the described tasks. An important part of the constructor is the registration of the *Application* at the corresponding *Host*. To ensure the *Application* to always know its current state, the enum *WSNWorkloadCycleState* defines the basic states `INIT, ACTIVE`, `SLEEP`, `ERROR`, `OFF`, `OTHER`. Thereby, by setting the state to `ON` or `OFF`, automatically actives and deactivates the *Host*. In the beginning, this state is set to `INIT`. This means the *Application* is not yet initialised. Due to the *Application* registered as an *Application* inside the constructor, the *Host* calls the *Application* to initialise after the simulation starts. The initialisation therefore takes place as soon as all components of the *Host* have been created. As shown in Listing 5.3 for the example of *EnergyAwareHostComponents*, lists are filled with the available components of a specific subtype of the *HostComponent* interface. Besides *EnergyAwareHostModules*, this is done for different transmission components like WiFi and LoRa components and for instances of the *WSNSensorComponent* interface. Although some components are part of multiple lists, this structure enables the application to quickly select the required components.

```
private List<EnergyAwareHostComponent> energyAwareComponents;
public void initialize() {
   // ...
   try {
      energyAwareComponents = getHost().getComponents(EnergyAwareHostComponent.class);
   }catch (ComponentNotAvailableException e) {
      // There are no EnergyAwareHostComponents available in this Host
   }
}
```

Listing 5.3: Extract of the initialisation of the *WSNWorkloadApplication* class. Instances of different host component subtype interfaces are stored in a list to enable the application so that the application can quickly access these components.

After the successful initialisation, the *Application* starts its timer by saving the current simulation time. Also, it generates an event, calling itself to start the workload cycle. The corresponding event handler checks whether the *Host* is still active and then calls the method `performWorkloadCycle()`. This method implements the central workload algorithm, that controls the tasks of the *Application* and thus also of the entire sensor node. For this reason, several helper methods are implemented. These are sub-routines for the execution of the sampling, transmission and sleep states as well as functional helper methods, for example for setting all *Host Components* to a specific state.

The basic structure of the method `performWorkloadCycle()` is a cascade of different queries, like whether the *Host* is still active and what is the current application state. As soon as a new task gets started, a delayed event is scheduled. Thereby, the delay time is set to the corresponding configured duration of this new task. This leads to this tasks is performed until the delayed event triggers the *Application* to switch to the next task. Since the duration variables are changeable during the runtime, this workload cycle stays dynamic in terms of the task durations. Listing 5.4 shows the event scheduling

for the wakeup after the sleep task. Since the workload cycle starts with the active tasks, the wakeup event restarts the `performWorkloadCycle()` method.

```
// Turn off all deactivatable components
setComponentsToState(WSNWorkloadCycleState.SLEEP);
if (isHostAlive()) {
   Event.scheduleWithDelay(sleepTime, new EventHandler() {
      @Override
      public void eventOccurred(Object content, int type) {
         // Wake-Up
         performWorkloadCycle();
      }
   }, null, 0);
}
```

Listing 5.4: Extract of the performWorkloadCycle method of the *WSNWorkloadApplication. The components are set to sleep. The timer is set to schedule the wakeup after the corresponding time.*

For scheduling the event which initiates the sleep phase at the end of the active time, the same structure is used. In addition, after setting the event, the active tasks sampling and transmission are started by calling the helper methods `startSampling()` and `startSending()`. Within these methods, events are again used to recall the methods itself after the specified time interval. The actual sample and transmission tasks are again outsourced to other methods. These fetch the required components from the corresponding initialised lists and set the states according to the current task.

To enable communication between multiple sensor nodes, the *Application* also provides a `messageArrived()` method, which is called as soon as an message is received from one of the registered transmission modules. With this method, *Applications* are able to react dynamically on different network activities. However, this behaviour is not implemented in this concrete *Application* component.

Since the *Application* manages and controls the tasks of the *Host Components*, some concrete implementations of different subtypes of the *HostComponent* interface are presented in the following.

Implementation of Configurable *Host Component* Subtypes for Generic Hardware Modules

Host Components are the central components, a *Host* consist of. These represent in particular models of hardware modules such as MCs, sensors or actuators. While the implementation of an *Application* just described cannot be assigned to a specific hardware module, the components shown here implement the functionality and energy consumption of specific hardware modules. This means, that the components described here, need to be supplied with sufficient energy. If this is not the case at a certain point in time, the sensor node is defined as inactive, due to its fully discharged *Battery*. Since there are plenty of different hardware modules, even with the same functionalities, generic models are used in this implementation. This means, that the components are configurable in terms of energy consumption and component type. These information are sufficient for this basic energy-aware simulation. However, detailed model descriptions and extended functionalities for concrete hardware modules can be implemented additionally in own classes.

In this work different configurable subtypes of the *HostComponent* interface are implemented. Looking at the configuration of three of these different *Host Components* in Listing 5.5, a certain basic similarity in terms of parameters and structure is recognisable. The number of parameters is quite manageable for all of them and these consist of a factory and at least one specification of energy consumption. The components *GenericWSNComponent* and *GenericWSNSensor* also provide a *componentType* or *sensorType* parameter. The *GenericMicrocontroller* component is parametrised with two consumption values: the energy consumption in the active state *energyConsumptionActiveUwatt*, and the energy consumption in the

sleep state *energyConsumptionSleepUwatt*. As decided in the beginning of this Chapter, both energy consumptions are technically power consumptions as they are defined in micro watts. Nevertheless, based on these values, the constructor of the *GenericMicrocontroller* class creates corresponding *EnergyState* objects, that define an energy consumption per second for a certain state in the *EnergyComponent*. Valid values for the *componentType* parameter are: *COMMUNICATION*, *POSITIONING*, *CPU*, *DISPLAY*, *BASIC*, *ACTUATOR*, *SENSOR* and *MICROCONTROLLER*. The *sensorType* parameter accepts a expandable enumeration of different sensor type names like *AIRQUALITY*, *TEMPERATURE* or *HUMIDITY*. This specification allows an *Application* to treat sensors differently. Even though this is not the case with the configurable *Application* shown here, this decision allows for a variety of other *Applications* further flexibility.

```
<!-- Active: 65mA * 3.7V = 240,500uW ;  Sleep: 18mA * 3.7V = 66,600uW-->
<GenericMicrocontroller class="(...).GenericMicrocontroller$Factory"
   energyConsumptionActiveUwatt="240500"
   energyConsumptionSleepUwatt="66600"
/>

<!-- 10mA * 3.7V = 37.0mW = 37,000uW -->
<GenericWSNComponent class="de.tud.kom.p2psim.impl.wsn.GenericWSNComponent$Factory"
   componentType="DISPLAY"
   energyConsumptionUwatt="37000"
/>

<!-- 33mA * 3.7V = 122.1mW = 122,100uW -->
<GenericWSNSensor class="de.tud.kom.p2psim.impl.wsn.GenericWSNSensor$Factory"
   sensorType="AIRQUALITY"
   energyConsumptionUwatt="122100"
/>
```

Listing 5.5: Configuration of different generic *HostComponent* subtypes.

All of these generic components always implement the *EnergyAwareHostComponent* interface, due to these components are modelled to consume energy. Depending on the actual component type, other interfaces like the *WSNSensorComponent* are also implemented in addition. The class *GenericWSNSensor* is a special case of a *GenericWSNComponent*, as it brings further sensor specific methods like specified in Section 5.2.1. Nevertheless, since the basic concept is the same, the *GenericWSNSensor* class extends the *GenericWSNComponent* class.

Different than the implementation of the *GenericWSNComponent* and the *GenericWSNSensor*, the implementation of the *GenericMicrocontroller* supports the use and the configuration for multiple *Energy States*. This can be seen in the configuration in Listing 5.5, where two energy consumption parameters are given for the *GenericMicrocontroller* component. Listing 5.6 shows the creation of two *EnergyStates* based on these consumption values. Further, the *MultistateEnergyComponent* is used to handle the created list of *EnergyStates*. The next sections goes in more detail, how this *MultiStateEnergyComponent* works.

MultiStateEnergyComponent

The *MultiStateEnergyComponent* is an implementation for an *Energy Component* that handles more than one *Energy State*. *Energy States* define one concrete energy consumption, mostly for one task inside an *Energy Component*. The task of the *Energy Component* is to handle these *Energy States* and so calculate the current energy consumption of the corresponding *Host Component*.

Since the *Simonstrator* already brings the *OneStateEnergyComponent* managing the energy consumption for *Host Components* with just one *Energy State*, the *MultiStateEnergyComponent* is added to extend

```
// create the corresponding energy states
EnergyState stateActive = new DefaultEnergyState("active", energyConsumptionActive);
EnergyState stateSleep = new DefaultEnergyState("sleep", energyConsumptionSleep);

// define a list for all states and add these
List<EnergyState> states = new LinkedList<>();
states.add(stateActive);
states.add(stateSleep);

// create the corresponding energy component
this.energyComponent = new MultiStateEnergyComponent(states, ComponentType.MICROCONTROLLER);
```

Listing 5.6: Extract from the *GenericMicrocontroller* constructor. It shows the creation and handling of multiple energy states.

the simulation environment with a more powerful *Energy Component*. As its name already suggests, the *MultiStateEnergyComponent* is able to calculate the energy consumption for more complex modules that consist of multiple different states with a different amount of energy consumption.

For this purpose, the *MultiStateEnergyComponent* gets constructed with a list of *Energy States* and the *Component Type* of the corresponding *Host Component*. Also, the local variable *currentState* is initialized with a new created default state with an energy consumption of 0 µW. This variable saves the current state, that can be changed by using the `public boolean changeState(EnergyState nextState)` method. This adds the provided *Energy State* to the list, if it is not already registered and sets it as the active state. Due to this *Energy Component* implements *Hybrid Pushing*, as explained in Section 4.2.2, after a successful change of state, the *Energy Component* consumes energy by notifying the *Energy Model*. For this purpose, the `changeState()` method schedules an event for itself without any delay but with the id parameter set to −1. This event calls the `eventOccurred(Object content, int id)` method which is shown in Listing 5.7. Since *Hybrid Pushing* also requires a static query interval, the method in addition schedules a new event with a delay of one second for itself, if the recent call was due to the static query interval. That is why the dynamic state change sets the id −1 at the event creation, since in this case, no new delayed event should be scheduled.

In order to simplify the handling of several states for application components, the *MultistateEnergyComponent* provides two methods for selecting an active and a sleep state. These methods try to search the list containing all known *Energy States* for the first state with the string "active" or "sleep" in its name field. This state is returned and can be used to easily control components with just one active and sleep state. This approach does not introduce any new parameters and is therefore compatible with previous configured *Simonstrator* simulations.

Otherwise, this class mainly consists of getter and setter methods, as well as the methods required by the implemented *EnergyComponent* interface for switching this component on and off.

ComponentBasedEnergyModel and Battery

The class *ComponentBasedEnergyModel* as well as the *SimpleBattery* class were almost fully implemented in the *Simonstrator* framework. However, since these classes are very important for this implementation, the main features are explained in this section.

As seen in the section before, the *Energy Model* gets notified from the corresponding *Energy Components* regularly, if energy is consumed by a *Host Component*. The called method `componentConsumedEnergy(...)` then manages the further procedure. This includes discharging the *Battery* with a given amount of consumed energy and checking the new *Battery's* state of charge. If the *Battery* has still energy left, everything is fine and the components proceed in their individual tasks.

```
// id = -1 means this was an state change triggered consumption call
public void eventOccurred(Object content, int id) {
   if(isOn()) {
      // calculate elapsed time since last energy consumption
      long interval = Time.getCurrentTime() - this.lastConsumption; // time in us
      double intervalS = ((double) interval) / Time.SECOND; // time in s
      // calculate the real energy consumption
      double consumedEnergy = currentState.getEnergyConsumption() * intervalS;

      // actions to perform, if event occurred
      energyModel.componentConsumedEnergy(this, consumedEnergy);  // consume energy
      this.lastConsumption = Time.getCurrentTime(); // update time of last consumption

      if (energyModel.componentCanBeActivated(this)) {
         if(id != -1){ // -1 means dynamic push due to state change
            // schedule next static query interval
            Event.scheduleWithDelay(Time.SECOND, this, null, 0);
         }
      } else {
         this.turnOff(); // deactivate component
      }
   }
}
```

Listing 5.7: The *eventOccured()* method from the *MultiStateEnergyComponent* class. Due to the use of *Hybrid Pushing*, his method gets called every second by itself, and after each state change.

If the *Battery* is now completely discharged, the *Energy Model* deactivates the *Host*, disconnects the transmission components and turns off all other *Host Components* registered at the *Energy Model*.

This behaviour is compatible with different types of battery models, as soon as these implement the *Battery* interface and provide a method to return the current state of charge. However, in this work, the already existing *SimpleBattery* class is used. This class does not include any kind of non-ideal battery behaviour. Instead, it is initialised with an specific amount of usable energy, that gets reduced over time by the components consuming energy. This model is not particularly realistic, but it has the advantage that it can be configured quickly and is easy to understand. To simulate scenarios in detail, the given energy capacity of the battery must therefore be manually adjusted to include non-idealities, or an extended battery model must be used.

5.3 Interaction of Components

Due to the modular design and the component-based approach of this implementation, multiple components interact with each other. Further, the integration of the new implemented model in the *Simonstrator* is arranged. This section explains the main principle and interfaces for the component interaction.

As already explained, the *Host* represents the central element of a sensor node. All *Host Components* register at the *Host* before the simulation starts. Thereby, the *Host* creates lists of different components implementing subtypes the *HostComponent* interface. With the associated getter methods, it is therefore possible to access specific components of a sensor node via the *Host*. On the other side, the *Host Components* get constructed with information about their corresponding *Host*. The *HostComponent* interface also forces the *Host Components* to implement a public getter method returning their *Host*. This makes it possible to access all other *Host Components* within a sensor node from each *Host Component* via the associated *Host*. Especially for the *Application*, that is also just another *HostComponent* interface subtype,

this access is necessary to control the other *Host Components*. The same applies to the *Energy Model*. This makes the *Host* an intermediary for interactions between *Host Components*.

In addition, it is possible to interact with the own component or with other components via the event-based interaction methods. These differ from the classic access via the *Host* in that they can be equipped with a time delay. Further, every event-based trigger, is handled as a parallel task. This means that the calling function does not have to wait until the called function is finished, but can continue immediately. However, it is therefore also not possible to get a return value. That is why these event-based component interactions are mostly used as an event notification, as already described in Section 5.2.2. This approach allows the simulation to process different tasks in parallel and to schedule events with a delay, managed by the simulation time interface.

5.4 Limitations of the Implementation

This implementation has been done primarily to show that the design presented is feasible. However, by no means all the possibilities of the design presented have been implemented and utilised. This is especially noticeable in the implementation of different *Host Components* for example the generic component types, which are mainly implemented as energy consumers but come with a limited logical functional implementation. If not only configurable energy behaviour is required for a simulation, but also extended logical behaviour, this implementation requires further adjustments. Due to the modular design and the *Host Components* having access to all other *Host Components* inside the sensor node, extended logical behaviour can be implemented without further adjustments to the simulation environment. However, due to the many possibilities of interaction between different *Host Components*, further development of logical functionality could even lead to some components interacting with other components in a way that is not supported by existing physically hardware modules. This may allow scenarios to be simulated that cannot be realised in hardware, which can represent both a chance and a risk.

Further, the used battery model is a highly simplified and idealised battery model. It does not consider any non-idealities and allows the components to consume as much energy as desired, as long as the *Battery* still has available energy left. Effects like *Peukert's Law* or the *Recovery-Effect* are also not implemented. Using this simplified *Battery* severely limits the precision of the simulation. However, the design used offers the possibility of expanding the simulation to include more precise, possibly hardware-specific battery models. An evaluation of the simulation, especially with regard to the achievable precision using the simplified battery model, is provided in Chapter 6.

5.5 Summary

In general, the presented implementation is very close to the proposed design in terms of the basic structure. By embedding further interfaces and classes, it is achieved that the *Simonstrator* framework is extended by the proposed model. In addition to the realisation of the basic structure of the proposed model, a particular focus is on the implementation of generic and configurable sensor node components, such as the *WSNSensorApplication* or the *GenericWSNComponent*. Also, the *MultiStateEnergyComponent* is introduced as a powerful *Energy Component*, that is able to manage the energy consumption for more complex *Host Components*. Further, since this modular approach needs different components to interact with each other, Section 5.3 explains how the functionality of the hosts is guaranteed regardless of the current sensor node configurations. Therefore, the host-centred architecture is pointed out.

Finally, Section 5.4 discusses the different limitations of this concrete implementation. These are mainly caused by the use of a simplified battery model and the lack of logically detailed modelled hardware modules. Among other things, it is also questioned to what extent this modular model with the use of many generic components is able to provide precise information about the lifetime of sensor nodes. Therefore the next chapter evaluates this aspect by using a proof-of-concept evaluation with real-world measurements.

6 Evaluation

After implementing the proposed design of a modular sensor node simulation model, this chapter evaluates this implementation. Thereby, Section 6.1 describes the goal and the methodology of this evaluation. Section 6.2 explains the therefore used evaluation setup, including the configuration of the simulation as well as the setup for the real-world measurements. In Section 6.3 the results of the measurements and the simulations are presented. Finally, in Section 6.4 the evaluation results are analysed in the overall context of this work.

6.1 Goal and Methodology

This evaluation is done as a *proof-of-concept* evaluation. This means, using the described implementation, the presented design is used to simulate a Wireless Sensor Network (WSN) scenario, considering the lifetime of different sensor node configurations. For this purpose, the battery-related lifetime of each simulated sensor node is evaluated individually and compared with a real-world measurement. Therefore, real hardware sensor node configurations are conceived and implemented in the simulation framework. Starting with fully charged batteries, the lifetimes of the sensor nodes are then evaluated and compared with the simulated results. Thereby, it is desirable that the simulated results come as close as possible to the values of the real-world data.

The goal of this evaluation is to proof the fundamental functionality and feasibility of the proposed concept. Further, the precision of the described implementation is analysed in terms of sensor node lifetime prediction to assess the benefit of this specific implementation of the model. It also examines how flexible and adaptable the model really is due to its modular structure.

Therefore, three different scenarios are determined that are simulated in the following as well as programmed and assembled in real hardware. All scenarios are based on the *Heltec ESP32 LoRa*[1] Microcontroller (MC) and the *CJMCU-811*[2] airquality sensor. Table 6.1 shows a detailed specification of the three scenarios. Since the sensor nodes in Scenario 1 are always in active state and do not turn into sleep mode, the active duration is set to infinity.

Scenario	Active Duration [s]	Sleep Duration [s]	Transmission Interval [s]	Sample Interval [s]	OLED Display
1	∞	0	10	0.25	ON
2	600	300	10	0.25	ON
3	600	300	10	0.25	OFF

Table 6.1: Defined application workload parameters for the real-world sensor node measurement and the related simulation. Transmission Interval and Sample Interval are defined within the active state.

By comparing the lifetimes of the real-world measurement with the simulated sensor node lifetimes, it is possible to evaluate the precision of the simulation in different scenarios. In addition to the basic simulation based on the nominal values of the used batteries, a second simulation series is performed based on the measured real-world data, which is configured with modified battery values. In order to be able to understand exactly how these measurements and simulations are implemented and performed, the used setups and configurations are explained in the following section.

[1] https://heltec.org/project/wifi-lora-32/

[2] https://www.makershop.de/download/CCS811_Datasheet-DS000459.pdf

6.2 Evaluation Setup

For the *proof-of-concept* evaluation of the proposed model, different real-world measurements are performed. First, the current consumptions and supply voltages of the individual hardware components need to be analysed. The resulting power consumption values are then used as parameters for the configurable *Host Components* inside the simulation model. Further, the real-world hardware sensor nodes have to be configured and programmed, so that these and the corresponding sensor nodes in the simulation perform the same tasks. In addition, a device for measuring the lifetime of the sensor node devices must be built. The collected data is then evaluated after all sensor nodes' batteries are fully discharged.

6.2.1 Acquiring Real-World Power Consumption Values for the Used Hardware Modules

To enable the simulation to be configured with suitable power consumption values for the individual configurable *Host Components*, the power consumption of all corresponding hardware modules have to be determined. For this reason, an Instrumentation Amplifier (INA)-based power, current and voltage measurement shield *TI INA219*[3] is used together with a custom made measurement program running on the connected *Heltec ESP32 LoRa* MC. This measurement setup is connected in series with the Device Under Test (DUT), as it is explained in Chapter 4.3. Sampling four consumption samples per second, consisting of current, voltage and power values, this measurement setup provides *CSV*-formatted consumption data via the MCs serial interface. Since the *Heltec ESP32 LoRa* is used as the MC for the evaluated sensor node configuration, different modules are already included in this hardware component. This makes the measurement more difficult, due to the individual hardware modules cannot be measured in isolation. Instead, comparative measurements are done by using an application that switches a specific hardware module on and off. This makes it possible to analyse the current consumptions of the OLED and LoRa module inside the *Heltec ESP32 LoRa* MC. Also the sleep state consumption can be measured by programming the MC to wake up and sleep regularly. In order to have a uniform measurement setup, the used air-quality sensor module *CJMCU-811* is also evaluated as a comparative measurement.

6.2.2 Evaluation Setup for Real-World Sensor Node Lifetime Measurements

For the real-world sensor node lifetime measurement the chosen hardware modules are the *Heltec ESP32 LoRa* development board including an OLED display, a LoRa transmission module and the ESP32 MC, together with the *CJMCU-811* air-quality sensor and a 3000 mA h rechargeable lithium battery. To enable the hardware to execute the correct tasks, the MC is programmed to perform the workload defined for the corresponding scenario. Since measuring the runtime of a sensor node requires equipping it with a fully charged battery and then waiting until the battery is fully discharged, many hours are needed to perform a measurement, depending on the application tasks. That's why, for this evaluation, three completely similar hardware configurations are set up and evaluated synchronously. This allows three measurements of the same hardware and application configuration to be taken, so that the significance of the measurement can be increased by calculating the average value. This resulting average is then used to compare the simulated lifetime with.

As shown in Table 6.1, three different scenarios are examined to consider different hardware modules being active and different application cycles are performed. *Scenario 1* samples sensor data every 250 ms and transmits a sample every 10 s. The currently measured samples are also shown on the OLED display. *Scenario 2* performs the same workload than *Scenario 1*, but adds a sleep phase of 5 minutes after being active for 10 minutes. *Scenario 3* performs the same workload than *Scenario 2*, but with the OLED display turned off. The average power consumptions of these scenarios therefore differ, due to the sleep phase and the switched-off display. One advantage of these three chosen scenarios is that they can all use the

[3] `https://www.ti.com/product/INA219`

same hardware setup and still allow different application workloads and hardware module usage. This allows the versatility of the model to be tested with regard to the modularity of the simulation model as well as the usability of the configurable sensor node application.

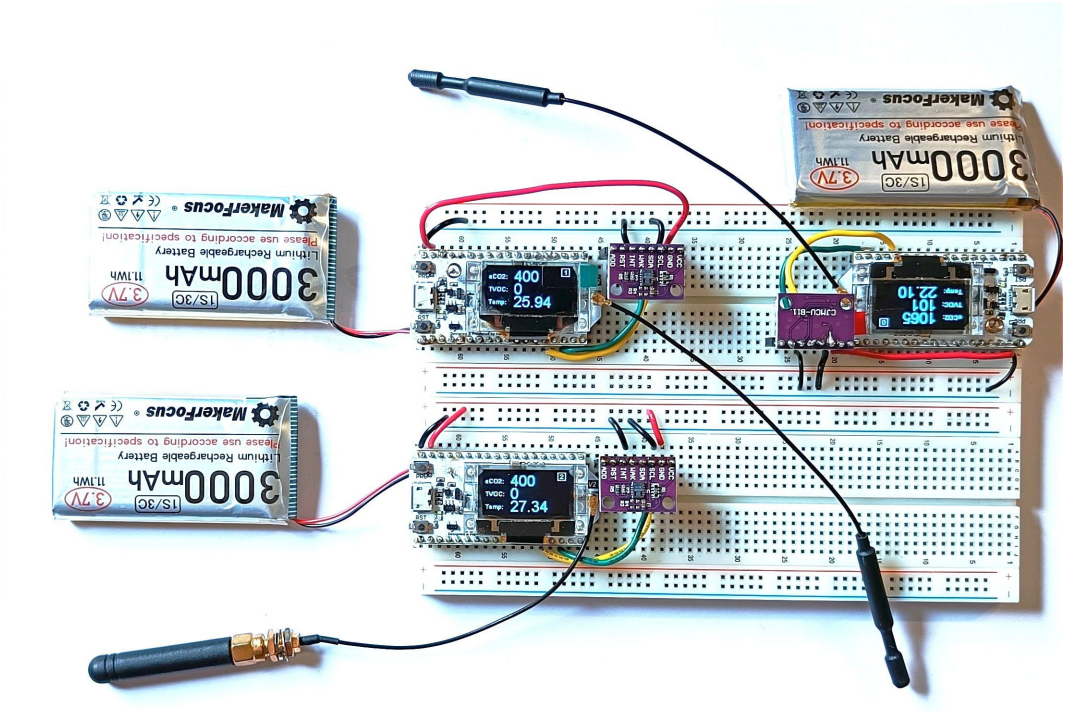
To ensure to correct lifetime measurement of the DUT, a measuring instance is built to receive and log the transmitted data from the sensor nodes. This is done be using another *Heltec ESP32 LoRa* together with a SD-card breakout board. Due to the on-board WiFi capabilities of the *ESP32*, the MC is able to initialize its internal time module with the current date and time via the internet. Using the transmitted individual device IDs, the receiver is able to log all incoming LoRa transmissions to device specific *CSV* files. In addition to the received data, the date and time of receipt as well as the corresponding Received Signal Strength Indicator (RSSI) value is also logged. This makes it possible to analyse the received and logged data from the SD card to determine which device went offline at which point in time and stopped transmitting. The hardware measurement setup including three devices under test and the receiver and logger station, is shown in Figure 6.1. Since the receiver and logger must not run out of battery, it is connected to the mains and buffered via a powerbank.

A major advantage of measuring via the receiver module is that a measurement can be taken without any measurement feedback on the sensor nodes. This enables an unbiased measurement. In addition, the receiver measures exactly the parameter that is relevant for a WSN sensor node, namely how long data transmission is still possible. Other measurement methods, such as measuring the battery voltage, cause direct measurement feedback, as they cause an additional current flow due to the measurement. Also, the MC's behaviour at low supply voltage would have to be analysed. These additional measurements can be avoided by using the presented measurement setup.

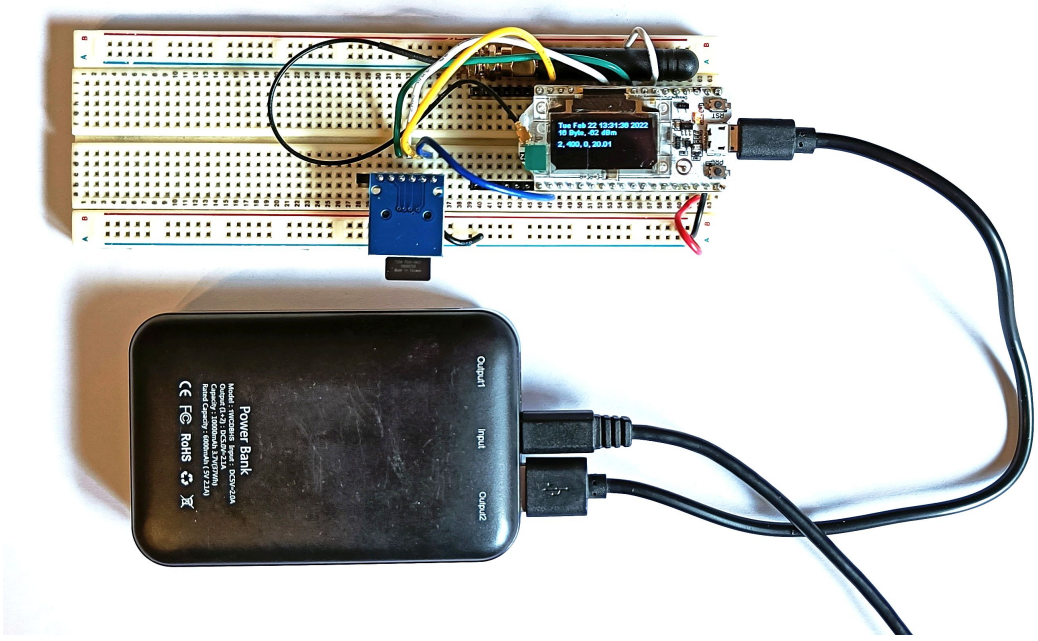
6.2.3 Sensor Node Configurations for the Simulation Execution

For the evaluation of the proposed model, two simulation series are configured. Both require the results of the consumption data measured in Section 6.2.1 since the chosen hardware modules are configured in the simulation using the generic components *GenericWSNComponent*, *GenericMicrocontroller* and *GenericWSNSensor*. The workload defined in Table 6.1 is implemented by configuring the *WSNSensorApplication* component with the respective durations for active and sleep states as well as the transmission and sampling intervals. The simplified battery model *SimpleBattery* is used as the battery model. For the first simulation the battery capacities are set to the batteries nominal capacity of 3000 mA h.

While all other parameters remain the same as in the first simulation, the battery capacities are adjusted in the second simulation which estimates the performance of the presented model using a perfect non-ideal battery implementation. For this purpose, the results from the real-world sensor node lifetime measurement are analysed. With the help of the number of successful data transfers and the consumption values of the individual modules and tasks, it is possible to calculate the energy that has been applied over the entire lifetime of a sensor node. In this way, it is possible to reverse engineer the energy that has been extracted from the battery. For this purpose, the duration of the sensor node in sleep and active state are calculated by multiplying the total lifetime with the active to sleep ratio. These two time values are then multiplied with the total current consumption of their respective state. The results in mA h are then added and represent the base energy consumption. On top comes the energy consumption of the transmission. For this purpose the number of received samples are multiplied with the duration of a transmission, which is here set to one second. The result represents the total transmission time and is multiplied with the transmission current, which leads to another consumption in mA h. Adding this consumption to the base consumption represents the total consumption of the sensor node in mA h which is set to the sensor nodes battery capacity. With these two simulation series, the upper and lower limits of the battery-related accuracy of the presented model can be simulated.



(a) Three equally configured hardware sensor nodes under evaluation in *Scenario 1*: *Heltec ESP32 LoRa* including a LoRa transmission component and the OLED turned ON, *CJMCU-811* airquality sensor, powered by *Makerfocus* 3000 mA h batteries, no sleep. The OLED displays the current sensor data as well as the sensor nodes individual node ID.



(b) Receiving station and logger: is used for all scenarios, logs the incoming LoRa signals on an SD card, is connected to the mains and buffered via a powerbank. The OLED displays the latest incoming connection including its daytime, RSSI, ID and sensor data.

Figure 6.1: Hardware setup for real-world data measurement of sensor node device lifetimes.

6.3 Evaluation Results

In the last section two measurement setups are shown. For the power consumption of the hardware modules which are used for the simulation of the sensor node lifetimes, a setup using an INA together with an MC is shown. Also, the setup with which the real-world sensor node lifetime measurements are done, is explained. This section now presents the collected data from the simulated results, as well as the measured lifetimes of the real-world sensor node lifetime measurements.

6.3.1 Measured Power Consumptions for Precise Sensor Node Component Configuration

For the simulation of the sensor nodes lifetimes, the consumption values of the corresponding hardware modules are required for the configuration of the sensor node components' individual *Energy Components*. These consumption values are determined by the measurement described in Section 6.2.1. The resulting data for the current consumptions of these different application tasks with different used hardware modules are shown in Figure 6.2.

Since comparative measurements were made, the *IDLE* measurement of the MC is particularly important, as all other measured data, except for the sleep measurement, refer to this value. As it can be seen from the *IDLE* line and in the spaces between the transmissions of the *LoRa* line in Figure 6.2, the current consumption of the switched-on MC is 65 mA in the IDLE state. For the sleep state, the MC requires an average of about 18 mA. The other consumptions for the air-quality sensor, the OLED and the LoRa transmission module must be read in relation to the idle state. Thus, the air-quality sensor comes to 33 mA and the LoRa module to about 70 mA while transmitting with spreading factor 7 and 7 dBm transmission power. For the OLED, the values vary greatly depending on the comparative measurement, the average value of the consumption is about 10 mA.

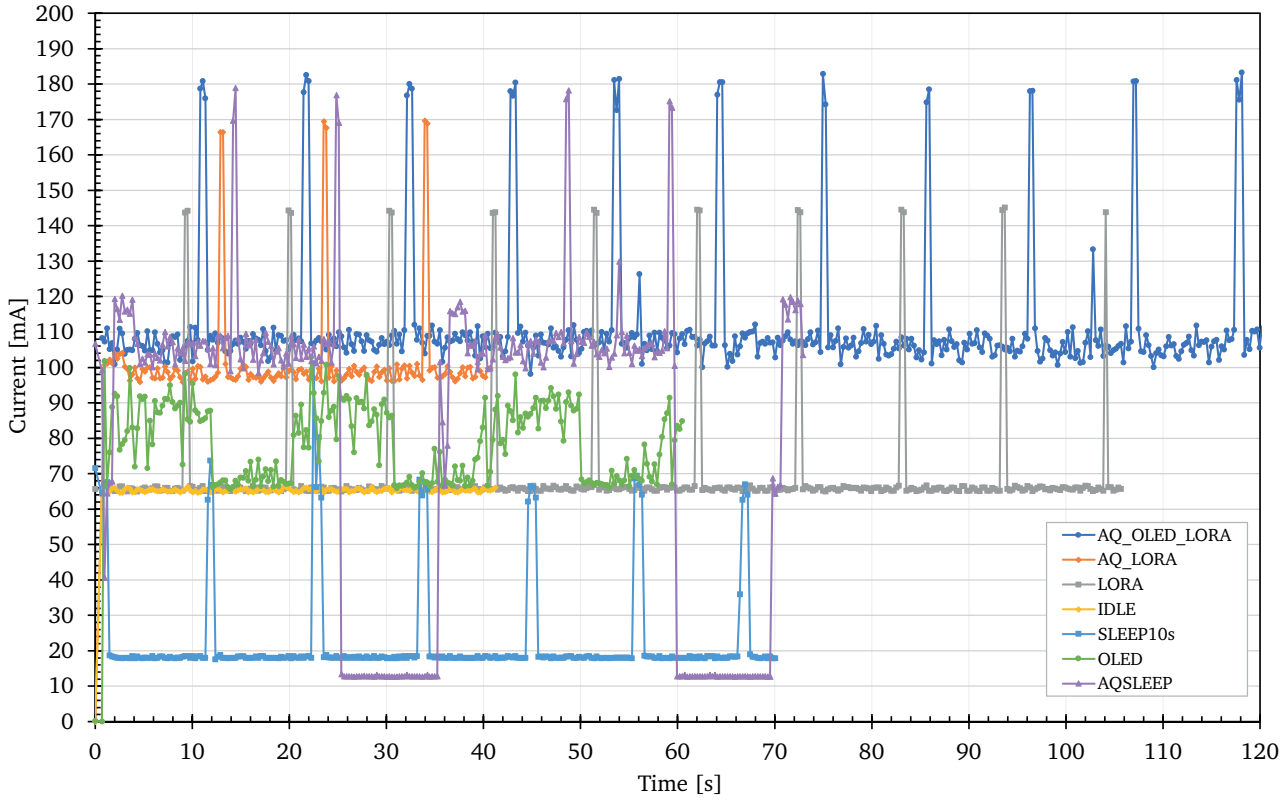


Figure 6.2: Current consumption of different application tasks analysed over time. The error for DC current measurements with the *TI INA219* is lower than ±0.5%.

The measurements took place at room temperature. The accuracy of the DC current measurement using the *TI INA219* is given with a typical error of ±0.2% and a maximal error of ±0.5%. Since the focus of this evaluation is not on precise measurement but rather on proof-of-concept, this order of magnitude for the measurement error is sufficient. These values are taken for the nodes' *Host Component* configurations which form the basis of the according simulation setup.

6.3.2 Collected Real-World Sensor Node Lifetime Data

As described in the previous section, three different sensor node scenarios are evaluated. The according measurements are done with three equal hardware setups parallel, which enables to collect more data simultaneously. For each sensor node all outgoing transmissions are logged together with the corresponding RSSI, date and time. Figure 6.3 shows the raw RSSI data for all three nodes and all three scenarios. However, the concrete value for the RSSI is less interesting, than the duration between the first and the last transmission of a sensor node. While *Scenario 1* still provides valid values for all sensor nodes, in *Scenario 2* and *Scenario 3* one or two sensor nodes fail after less than four hours of operation. The respective sensor nodes did not wake up from their sleep phase and therefore did not transmit any more samples. Since all three hardware configurations are built exactly the same and use the same software, it is not apparent what caused these early failures in some sensor nodes. However, an early failure of the batteries can be excluded, as the associated batteries still provided a voltage of more than the nominal voltage of 3.7 V. The initial battery voltage at the beginning of the measurement is 4.2 V. As these early sensor node failures are therefore not energy-related, they are not taken into account in the further evaluations. However, this also means that less data is available for *Scenarios 2* and *3*, so that the significance of these data series is also lower than for *Scenario 1*.

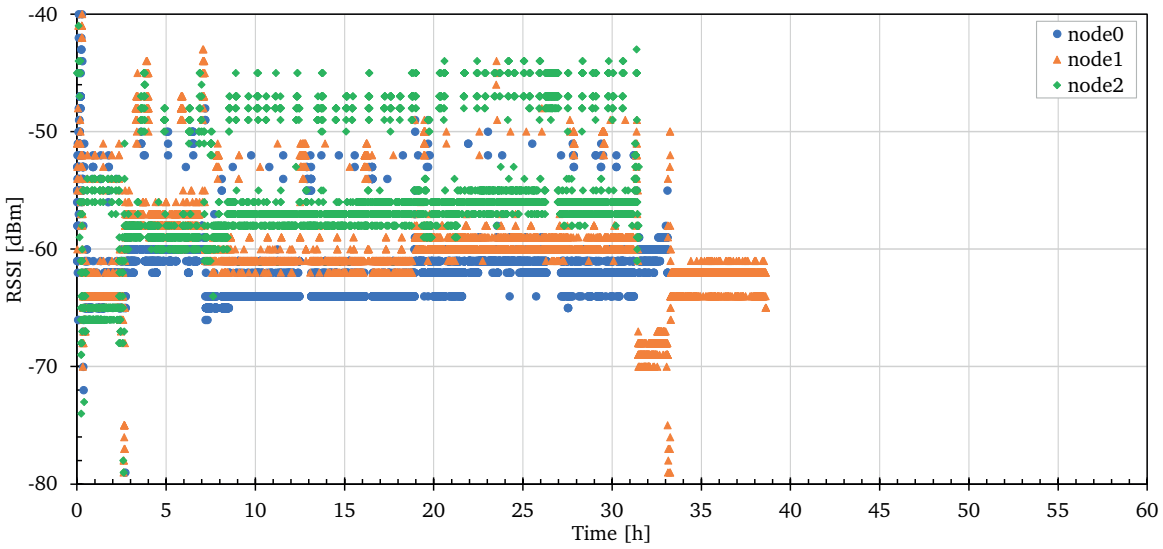
Further, it is also apparent that the lifetimes of the other sensor nodes differ greatly in some cases. For example in *Scenario 1*, as shown in Figure 6.3a, *node2* has a lifetime of 31:31h. In the same scenario, *node1* has a lifetime of 38:37h, which is an difference of 7:06h (18.4 %). Using Listing 6.1 to calculate the standard deviation s, the relative standard deviation is calculated by dividing s trough the average value μ.

$$s = \sqrt{\frac{\Sigma_{i=1}^{n}(x_i - \mu)^2}{n-1}} \tag{6.1}$$

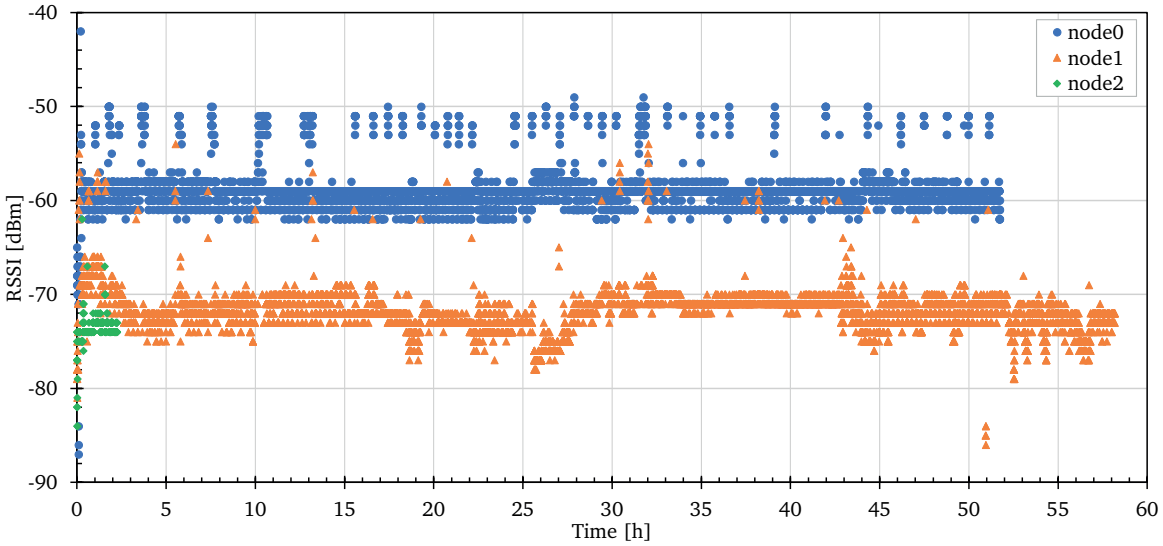
Table 6.2 shows that for the scenarios with several valid measurement data, high relative standard deviations of more than 10 % are present. This high fluctuation can best be explained by a fluctuation in the battery capacity. This means that the same battery modules can deliver different amounts of energy, even though they are from the same model and have the same nominal capacity of 3000 mA h.

	Scenario 1	**Scenario 2**	**Scenario 3**
# valid measurements	3	2	1
average value μ [h:m]	34:25	56:02	54:36
standard deviation s [h:m]	3:43	7:05	/
relative standard deviation s/μ	10.8%	12.6%	/

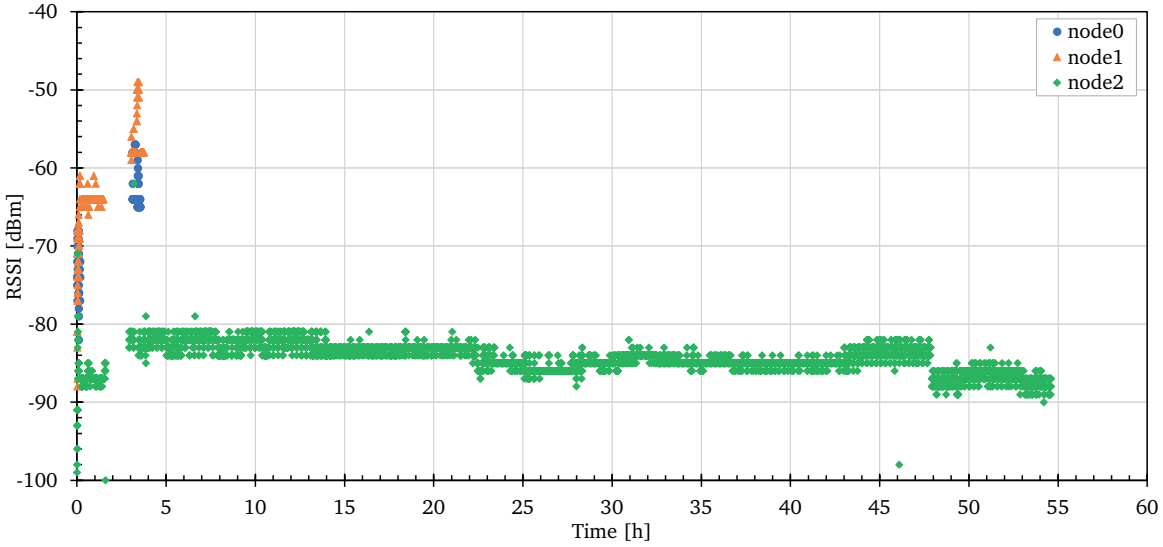
Table 6.2: Overview over the average lifetime value, standard deviation and relative standard deviation for all three scenarios. Since *Scenario 3* consists of just one valid measurement, no deviations are calculated.



(a) Scenario 1: 0% sleep, OLED ON



(b) Scenario 2: 33.3% sleep, OLED ON (ERROR on node3 after 2.5 hours)



(c) Scenario 3: 33.3% sleep, OLED OFF (ERROR on node1 and node2 after 3.5 hours)

Figure 6.3: RSSI measurements over time of three different sensor node configuration scenarios.

6.3.3 Simulated Sensor Node Lifetime Results

The core of the evaluation is the lifetime of the sensor nodes within the simulated WSN. In order to be able to classify these, real-world measurements of the exact sensor node compositions were made, as described in the previous section. Therefore, the configuration of the simulation is matched with the real-world measurement as shown in Section 6.2.3. For the basic simulation, the resulting lifetimes of the sensor nodes are 26:37h, 37:16h and 40:31h for the three scenarios, respectively.

As described in Section 6.2.3, a second simulation series is performed. Thereby, the used battery capacities are based on the results of the real-world sensor node lifetime measurements. For *Scenario 1* the average calculated used energy is 3636.52 mA h which is equal to 48.44 J by assuming a constant voltage of 3.7 V. *Scenario 2* has a calculated used average battery capacity of 4358.79 mA h and *Scenario 3* has 4205.09 mA h. Based on these adjusted battery capacities, a simulation can now be carried out that corresponds to the use of an optimal battery model. With the results of this simulation series, it is possible to estimate approximately how good this implementation can be using a detailed battery model. The resulting values for the three scenarios using this adjusted simulation are 32:16h, 53:32h and 56:49h.

6.4 Analysis of Results

After presenting the results of the real-world measurement and simulation in the previous section, this section analyses the results in terms of accuracy and discusses how these results should be placed in the overall context of this work. For this, Figure 6.4 summarizes the evaluation results, including the average and minimum value of the real-world measurements, as well as the simulated lifetimes of the basic and adjusted simulation for each scenario. Further, information about the scenarios application workload, the number of valid results in the real-world measurement and the corresponding standard deviation are given as well for each scenario.

Already when looking at the individual results of the real-world measurements in Figure 6.4, it is noticeable that there are large variations in the sensor nodes lifetimes. Accordingly, the relative standard deviations are also high at over 10 % in each case. These large fluctuations therefore make it difficult to classify the precision of the simulation. A guideline for precision would therefore be whether the results of the simulation are within the standard deviation around the mean value of the measurements. On the other hand, the minimum runtime of a sensor node is of particular interest for many applications of such a simulation. With the median, one can assume that at this point in time about half of the sensor nodes are already switched off. However, if not much data is available, as in this evaluation, it also makes sense to look at the mean value, as this indicates the average duration of all measurements. All of these approaches are sensible approaches for the evaluation, in this case the deviation from the mean value is considered, as this corresponds most to the goal of the simulation to be as accurate as possible. For further evaluation, *Scenario 3* is neglected in the following, as the real-world measurement has only very limited significance due to the limited data available.

When looking at the results in Figure 6.4, the simulated lifetimes with the basic battery configuration with 3000 mA h is significantly smaller than the measured average value and even the measured minimum value. The relative deviation is 23.7 % and 33.5 %. Thus, the base value is also not in the range defined by the standard deviation around the mean value.

The adjusted simulation, which was configured with the back-calculated consumed energy, is located in the desired environment around the average. With a deviation of 6.2 % and 4.5 % to the average value of the measurements of the first two scenarios. It is interesting to note that in this case the energy actually consumed is higher than the theoretical nominal energy available in the battery. There are two main reasons for this. On the one hand, it could be that the newly purchased batteries for this experiment can deliver even more energy than specified. However, this does not explain the energy difference between the scenarios. A possible explanation for this would be the *Recovery-Effect* described in Chapter 3.1.1. Especially in *Scenario 2*, where the sensor nodes are one third of their time in the

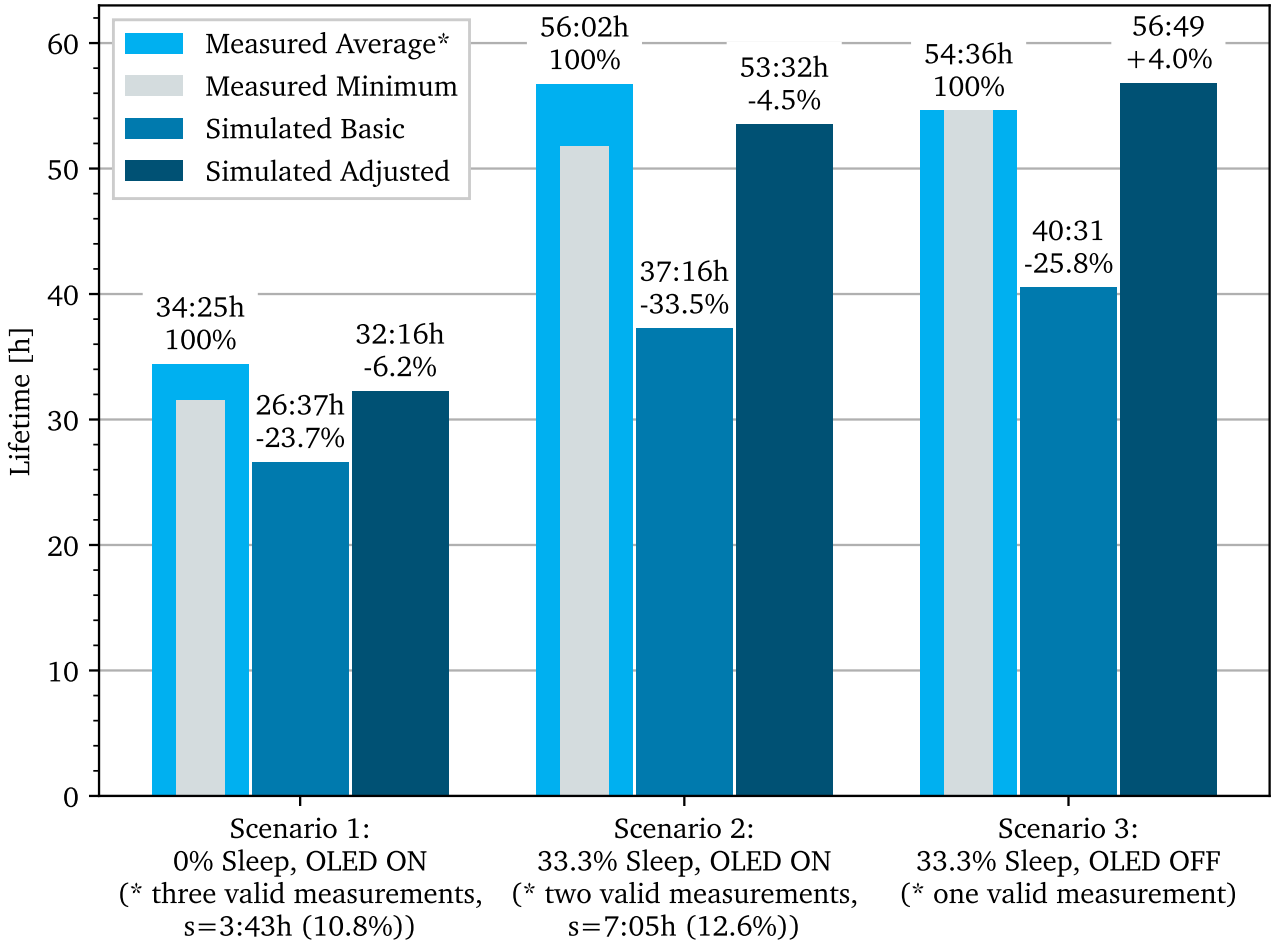


Figure 6.4: Evaluation results including the average and minimum value from the real-world measurements. In addition, the simulation results are given for the basic and the adjusted simulation. The percentage values indicate the relative deviation from the average value. For *Scenario 1* and *2* the standard deviation *s* is given as an absolute and relative value.

current-saving sleep mode, the calculated consumed energy with 4501.37 mA h, is significantly greater than in *Scenario 1* with 3636.52 mA h. This strongly suggests that the battery not only discharges more slowly due to the sleep phase with the lower power consumption, but that the voltage at the battery regenerates due to the *Recovery-Effect*.

Overall, this evaluation shows that this model with its modular structure is basically capable of simulating sensor nodes in a WSN. It is important to note that the actual lifetimes of the sensor nodes can vary greatly and this simulation only provides an average value, not a lower and upper bound. Furthermore, this evaluation has shown that it is necessary to use an extended battery model to get more accurate results from the simulation. The highly simplified battery model leads to a deviation of over 30 % in some scenarios. This may be sufficient for some applications, but this model offers the possibility to increase the precision by using extended battery models.

In addition to the analysis of the precision, it can also be said that the modular design allows a quick configuration of different sensor nodes. Especially through the use of configurable components, the simulations in this evaluation could be configured quickly and flexibly. The design and implementation presented here thus enable a modular configuration for the simulation of battery-powered sensor nodes in a WSN. Due to the modular design, this approach also allows the flexible use of different battery

models. These have an essential influence on the precision of the simulation and can be configured individually for different sensor nodes as desired.

After discussing the general design, presenting a concrete implementation and evaluating the model in terms of precision, the next chapter provides a summary about this work and its contribution. Also possibilities for future work based on this work are given.

7 Conclusions

In this Chapter, a final summary on the work presented in this paper is done. Section 7.1 summarizes the achievements of the main chapters of this paper. The main contributions of this work are shown in Section 7.2. Possibilities for future work based on this paper are discussed in Section 7.3.

7.1 Summary

This work presents a modular approach for energy-aware modelling of sensor nodes inside a Wireless Sensor Network (WSN). This requires a deep understanding of the structure and functioning of WSNs, which is given in Chapters 1 and 2. For the development of the sensor node model, its structure and the components it contains are of particular interest. As already examined in Chapter 3, the components of a sensor node can be clustered into sensors, batteries, transmission modules, Microcontrollers (MCs) and other hardware components. Due to the modular approach of this work, these models are all modelled as individual sensor node components. Since single sensor node components can be re-used for the configuration of other scenarios, this enables simulations based on this model to be highly flexibly. However, this also means, that every component, used in a specific scenario, needs to be modelled as an individual component. For this reason several related approaches are studied to gain further knowledge on energy consumption of typical sensor node modules, different battery models and other approaches for sensor node modelling and WSN simulations. Thereby, particular attention is paid to the different battery models, as these play a central role in the calculation of the sensor node lifetimes.

Chapter 4 then presents the concepts of the proposed modular energy-aware sensor node model. The focus here is heavily on the modular design. The presented sensor node model therefore allows an arbitrary amount of different *Sensor Node Components*. Accompanied by a *Battery*, an *Energy Model* and an *Application*, any number of hardware component models can be added as an *Sensor Node Component* to the sensor node. With associated *Energy Components* defining the power consumption of the components, the *Energy Model* is able to calculate the current energy demand and discharges the battery accordingly. For the interaction of sensor node components, the *Application* is configured to define different workload cycles and phases, where the *Sensor Node Components* perform different tasks. Overall, this modular concept leads to individual components consuming energy, so that at some point the battery is discharged and the sensor node is no longer considered in the simulation. The time span between the starting point of a sensor node and its shutdown is returned by the simulation as the sensor nodes lifetime. Since these sensor nodes may differ in their specific configuration, this lifetime considerations are simulated individually for every sensor node.

The implementation of this design is available out of the box in the simulator *Simonstrator*, which is very suitable for this purpose. With a few exceptions in the field of the configuration of the *Energy Model* component, the design can be adopted directly. The modular approach is particularly effective due to the configuration option via an XML file structure, where all simulation parameters including all sensor node configurations are specified. Because of the use of generic configurable sensor node components, the integration of new hardware modules is very fast and easy.

As a part of the *proof-of-concept* evaluation, the shown implementation gets evaluated in Chapter 6. Comparing the simulated lifetime results with real-world measurements of the corresponding sensor node configurations, the precision of the model is examined. The evaluation shows that the model works and the lifetimes of the sensor nodes vary depending on the hardware and application workload used. Using a highly simplified ideal and linear battery model, the simulation results of the model deviate by about 30 % on average from the real-world measurements. Due to the time-consuming experiments, the evaluation provides a sparse amount of measured real-world data, which gives a general intuition of the real lifetimes of the sensor nodes. However, the accuracy of the model can be increased even

further by using extended battery models. Thus, the energy provided by the batteries in the real-world measurements are reverse engineered. The calculated values are then used to tune the available energy from the individual simple battery model instances of the corresponding sensor node configurations. In principle, this imitates a perfect battery model. This increases the accuracy significantly and sets the average deviation to around 5 % compared to the real-world measurements. This shows that the proposed model is able to simulate good reference values. With the use of accurate battery models, it is even possible to perform simulations with high precision. Overall, the evaluation shows that the basic concept of a modular sensor node model works.

7.2 Contributions

The main contribution of this work is the development of an highly configurable modular energy-aware model for WSN simulations. This enables a fast and easy to configure energy-aware simulation of individual lifetimes of sensor nodes within a WSN. One advantage of this approach compared to approaches that do not design sensor nodes in a modular way is that individual components can be composed as desired, like in a construction kit. This does not require any considerations of the sensor nodes total energy consumption, but reduces the necessary energy consumption specifications to the component level. This makes the reuse of singe components in different scenarios possible, which in the long run accelerates the configuration process due to the increasing number of available, already configured component models. Further, due to the modular design, the presented model is easily extendible by implementing new sensor node components. With this model, the achievable deviation of the simulated lifetimes compared to real-world measurements varies between 30 and 5 % depending on the accuracy of the battery model.

In addition to the provided model, the creation and use of configurable generic sensor node components is also shown, which further simplifies the configuration process for new components. Also, for the evaluation process, a simple measurement setup using the *TI INA219* shield for measuring the energy consumption of hardware components is shown.

7.3 Future Work

Since the implementation shown here is strongly limited in terms of precision by the simplified battery model used, it is worth developing a generic and configurable battery model for sensor nodes in WSNs that consider several not-ideal effects like the *Recovery-Effect*. As shown in the evaluation, this would enable an improvement in precision of up to 25 %. For an accurate evaluation and adjustment of the presented model, further real-world measurements of sensor node lifetimes would improve the precision and validity of the models accuracy. In addition, further measurements can be made with other types of hardware components to confirm the general applicability of the model. Lastly, since WSNs in certain scenarios consist of a very high amount of sensor nodes, a study on the models performance in terms of scalability of large WSNs is interesting and could result in further improvements of the presented model.

Bibliography

[AA11] Nor Azlina Ab. Aziz and Kamarulzaman Ab. Aziz. Managing disaster with wireless sensor networks. In *13th International Conference on Advanced Communication Technology (ICACT2011)*, pages 202–207, February 2011.

[AAIS14] Aqeel-ur-Rehman, Abu Zafar Abbasi, Noman Islam, and Zubair Ahmed Shaikh. A review of wireless sensors and networks' applications in agriculture. *Computer Standards & Interfaces*, 36(2):263–270, February 2014.

[AAT+18] Kofi Sarpong Adu-Manu, Nadir Adam, Cristiano Tapparello, Hoda Ayatollahi, and Wendi Heinzelman. Energy-Harvesting Wireless Sensor Networks (EH-WSNs): A Review. *ACM Transactions on Sensor Networks*, 14(2):10:1–10:50, April 2018.

[ALM05] Th. Arampatzis, J. Lygeros, and S. Manesis. A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. In *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.*, pages 719–724, June 2005.

[APS+09] Ch. Antonopoulos, A. Prayati, T. Stoyanova, C. Koulamas, and G. Papadopoulos. Experimental evaluation of a WSN platform power consumption. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, May 2009.

[ASM+11] A. A. Abdou, A. Shaw, A. Mason, A. Al-Shamma'a, J. Cullen, and S. Wylie. Electromagnetic (EM) wave propagation for the development of an underwater Wireless Sensor Network (WSN). In *2011 IEEE SENSORS*, pages 1571–1574, October 2011.

[BV08] Leonardo Barboni and Maurizio Valle. Experimental Analysis of Wireless Sensor Nodes Current Consumption. In *2008 Second International Conference on Sensor Technologies and Applications (Sensorcomm 2008)*, pages 401–406, August 2008.

[CR06] Min Chen and G.A. Rincon-Mora. Accurate electrical battery model capable of predicting runtime and I-V performance. *IEEE Transactions on Energy Conversion*, 21(2):504–511, June 2006.

[DNDNCS16] Attilio Di Nisio, Tommaso Di Noia, Carlo Guarnieri Calò Carducci, and Maurizio Spadavecchia. High Dynamic Range Power Consumption Measurement in Microcontroller-Based Applications. *IEEE Transactions on Instrumentation and Measurement*, 65(9):1968–1976, September 2016.

[HD13] Austin Hausmann and Christopher Depcik. Expanding the Peukert equation for battery capacity modeling through inclusion of a temperature dependency. *Journal of Power Sources*, 235:148–158, August 2013.

[HGL+21] Jonas Höchst, Jannis Gottwald, Patrick Lampe, Julian Zobel, Thomas Nauss, Ralf Steinmetz, and Bernd Freisleben. *tRackIT OS: Open-source Software for Reliable VHF Wildlife Tracking*. Gesellschaft für Informatik, Bonn, 2021.

[KCZ07] Baoqiang Kan, Li Cai, and Lei Zhao. An accurate energy model for wsn node and its optimal design. In *2007 International Conference on Communications, Circuits and Systems*, pages 328–332, July 2007.

[KGT09] JeongGil Ko, Tia Gao, and Andreas Terzis. Empirical study of a medical sensor application in an urban emergency department. In *Proceedings of the Fourth International Conference on Body Area Networks*, BodyNets '09, pages 1–8, Brussels, BEL, April 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[KK16] Ingemar Kaj and Victorien Konané. Modeling battery cells under discharge using kinetic and stochastic battery models. *Applied Mathematical Modelling*, 40(17):7901–7915, September 2016.

[KKNL06] V. Konstantakos, K. Kosmatopoulos, S. Nikolaidis, and T. Laopoulos. Measurement of Power Consumption in Digital Systems. *IEEE Transactions on Instrumentation and Measurement*, 55(5):1662–1670, October 2006.

[KNVK20] Dionisis Kandris, Christos Nakas, Dimitrios Vomvas, and Grigorios Koulouras. Applications of Wireless Sensor Networks: An Up-to-Date Survey. *Applied System Innovation*, 3(1):14, March 2020.

[KPA+10] Fotis Kerasiotis, Aggeliki Prayati, Christos Antonopoulos, Christos Koulamas, and George Papadopoulos. Battery Lifetime Prediction Model for a WSN Platform. In *2010 Fourth International Conference on Sensor Technologies and Applications*, pages 525–530, July 2010.

[KPKK19] Dmitriy N. Karlov, Yulia S. Polozhentseva, Lyudmila V. Kremleva, and Dilovar D. Kalimyllin. The implementation of the IoT concept in the post-industrial economy. *Revista ESPACIOS*, 40(38), November 2019.

[KPMB08] Simon Kellner, Mario Pink, Detlev Meier, and Erik-Oliver BlaB. Towards a Realistic Energy Model for Wireless Sensor Networks. In *2008 Fifth Annual Conference on Wireless on Demand Network Systems and Services*, pages 97–100, January 2008.

[KTK15] Priyanka Kakria, N. K. Tripathi, and Peerapong Kitipawang. A Real-Time Health Monitoring System for Remote Cardiac Patients Using Smartphone and Wearable Sensors. *International Journal of Telemedicine and Applications*, 2015:1, 2015.

[Lou21] Andre Louie. 35 IoT Device Statistics You Must Read: 2022 Data on Market Size, Adoption & Usage. https://financesonline.com/iot-device-statistics/, June 2021.

[LWG05] O. Landsiedel, K. Wehrle, and S. Gotz. Accurate prediction of power consumption in sensor networks. In *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, pages 37–44, May 2005.

[MBCM18] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 197–202, March 2018.

[MWHA09] Geoff V. Merrett, Neil M. White, Nick R. Harris, and Bashir M. Al-Hashimi. Energy-Aware Simulation for Wireless Sensor Networks. In *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1–8, June 2009.

[OROR18] Chinedu I. Ossai, Nagarajan Raghavan, Chinedu Ishiodu Ossai, and Nagarajan Raghavan. Stochastic Model for Lithium Ion Battery Lifecycle Prediction and Parametric Uncertainties. In *2018 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6, January 2018.

[Peu97] Wilhelm Peukert. Über die Abhängigkeit der Kapazität der Bleiakkumulatoren von der Stromstärke. *Elektrotechnische Zeitschrift ETZ*, 18:287–288, 1897.

[RAA+14] Habib F. Rashvand, Ali Abedi, Jose M. Alcaraz-Calero, Paul D. Mitchell, and Subhas Chandra Mukhopadhyay. Wireless Sensor Systems for Space and Extreme Environments: A Review. *IEEE Sensors Journal*, 14(11):3955–3970, November 2014.

[RMB+17] Leonardo M. Rodrigues, Carlos Montez, Gerson Budke, Francisco Vasques, and Paulo Portugal. Estimating the Lifetime of Wireless Sensor Network Nodes through the Use of Embedded Analytical Battery Models. *Journal of Sensor and Actuator Networks*, 6(2):8, June 2017.

[RSKN05] V. Rao, G. Singhal, A. Kumar, and N. Navet. Battery model for embedded systems. In *18th International Conference on VLSI Design Held Jointly with 4th International Conference on Embedded Systems Design*, pages 105–110, January 2005.

[RSRS15] Björn Richerzhagen, Dominik Stingl, Julius Ruckert, and Ralf Steinmetz. Simonstrator: Simulation and Prototyping Platform for Distributed Mobile Applications. In *The 8th EAI International Conference on Simulation Tools and Techniques (ACM SIMUTOOLS 2015)*, pages 99–108, August 2015.

[WXVS14] Chaonan Wang, Liudong Xing, Vinod M. Vokkarane, and Yan (Lindsay) Sun. Reliability and lifetime modeling of wireless sensor nodes. *Microelectronics Reliability*, 54(1):160–166, January 2014.