Investigation of Deployment-Specific Simulation for the Optimization of Wireless Sensor Networks

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Jan Moritz Strübe
geboren am 18.09.1982
in Hannover

Eingereicht am: 27.06.2017
Disputation am: 20.11.2017
1. Referent: Prof. Dr. Rüdiger Kapitza
2. Referent: Prof. Dr. Falko Dressler

2017

Contents

Zusammenfassung **vii**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Objectives . . . 4
1.3 Approach and Overview . . . 4
1.4 Structure . . . 5
1.5 Definitions and Clarifications . . . 6
1.5.1 Distinction: Native and Target Platform . . . 6
1.5.2 Clarification: Application . . . 6
1.5.3 Distinction: Deployment and Testbed . . . 6
1.5.4 Demarcation: Low-Power Wireless Sensor Networks . . . 6
1.5.5 Node Versus Mote . . . 7

2 Background and Context **9**
2.1 Wireless Sensor Networks . . . 9
2.1.1 Sensor Nodes . . . 11
2.1.1.1 Micro Controller . . . 11
2.1.1.2 Radio . . . 12
2.1.1.3 Sensors and Actors . . . 13
2.1.1.4 Other Components . . . 13
2.1.1.5 Nodes . . . 15
2.1.2 Network Metrics . . . 16
2.2 Challenges for WSNs . . . 16
2.2.1 Analyzing and Debugging WSNs . . . 17
2.3 Network Protocols . . . 19
2.4 Simulating WSNs . . . 20
2.4.1 Network Level Simulator . . . 21
2.4.2 Instruction Level Simulator . . . 21
2.4.3 System Level Simulator . . . 21
2.4.4 Other WSN Simulators . . . 22
2.5 WSN Simulator Design . . . 23
2.5.1 Hardware . . . 24
2.6 Low Power Radio Models . . . 24
2.6.1 Data Transmission: Bit Streams, Bytes and Packets . . . 25
2.6.2 Channel, Modulation and Encoding . . . 25
2.6.2.1 Modulation . . . 26
2.6.2.2 Encoding . . . 27
2.6.3 Example Radios . . . 28
2.6.4 External Noise Sources . . . 29
2.6.5 Transceiver Model . . . 29
2.6.5.1 Interacting Properties . . . 30
2.6.5.2 Receive Signal Strength Indicator (RSSI) and Link Quality Indication (LQI) 30
2.6.5.3 Radio-Chip Features . . . 30
2.6.6 Signal Propagation Model . . . 31
2.6.6.1 Directed Graph Model . . . 32
2.6.6.2 Unit Disk Graph Model . . . 32
2.6.6.3 Free Space Model . . . 32
2.6.6.4 Two Ray Model . . . 32

2.6.6.5 Log-Distance Path Loss Model . . . 33
2.6.6.6 Log-Normal Shadowing Model . . . 33
2.6.6.7 Raytracing . . . 33
2.6.6.8 Additional Effects . . . 33
2.6.6.9 Trace-Based Models . . . 34
2.6.7 Reception Model . . . 35
2.6.7.1 Noise . . . 35
2.6.7.2 Basic Models . . . 35
2.6.7.3 Signal-to-Interference-Plus-Noise Ratio . . . 35
2.6.7.4 SINR based Bit Error Rate and Packet Reception Ratio (PRR) . . . 36
2.6.7.5 Experimental Models . . . 36

3 Problem Analysis and Suggested Approach 39
3.1 Optimization Objectives . . . 40
3.2 Severity of the Optimization Problem . . . 42
3.2.1 Discussion: Documentation . . . 43
3.2.2 Changing Conditions . . . 44
3.3 Optimization Strategy . . . 44
3.4 Adjusting WSN Software . . . 45
3.4.1 Build-Time Variability . . . 45
3.4.1.1 Build System . . . 46
3.4.1.2 Adjusting Files . . . 46
3.4.1.3 Libraries . . . 48
3.4.1.4 Tools . . . 48
3.4.1.5 Software Revisions . . . 48
3.4.1.6 Tool Invocation . . . 48
3.4.2 Run-Time Variability . . . 48
3.5 Executing Experiments . . . 49
3.5.1 Evaluating WSNs Experiments and Its Limitations . . . 49
3.5.1.1 Acquiring Data . . . 49
3.5.1.2 Interpreting the Data . . . 51
3.5.2 Testing Configurations . . . 52
3.6 Realism in Simulation . . . 52
3.6.1 Monte Carlo Simulation and the Pseudo Random Number Generator (PRNG) . . . 53
3.6.2 Node Start-Up Order and Inter-Node Offsets . . . 54
3.6.3 Simulator Abstraction Level . . . 56
3.6.4 Digital Hardware . . . 57
3.6.5 Clock Skew and Drift . . . 58
3.6.6 Actors . . . 59
3.6.7 Sensory . . . 59
3.6.8 Network Model . . . 60
3.6.9 Radio Hardware . . . 61
3.6.10 Energy . . . 61
3.6.11 Discussion . . . 62
3.7 Analysis of the Background Noise . . . 63
3.7.1 Monitoring Application . . . 63
3.7.2 Analysis . . . 66
3.7.3 Discussion . . . 72
3.8 Suggested Approach: Trace-Based Mapping to Enable Simulation-Supported Optimization 73
3.8.1 Related Work . . . 73
3.8.2 Mapping Networks to Simulators . . . 74
3.8.3 Collecting Network Attributes and Their Limitations . . . 74
3.8.3.1 Packet Reception Ratio (PRR) . . . 74
3.8.3.2 Receive Signal Strength Indicator (RSSI) / Signal Strength . . . 75
3.8.3.3 Link Quality Indication (LQI) . . . 76
3.8.3.4 Delays / Time of Flight (TOF) . . . 76
3.8.3.5 Location . . . 76
3.8.4 Conclusion . . . 76

3.8.5 Outlook: Continuous Monitoring . . . 77

4 RealSim: Mapping Deployments to a Simulator 79
4.1 Survey Application . . . 80
4.1.1 Methodology . . . 80
4.1.2 Technical Implementation . . . 80
4.1.2.1 Initialization . . . 81
4.1.2.2 Sending Packets . . . 81
4.1.2.3 Receiving Packets . . . 82
4.1.2.4 Background Noise . . . 82
4.1.2.5 Data Output . . . 83
4.1.3 Processing the Captured Data . . . 83
4.1.4 Related Work . . . 83
4.2 Replaying Data . . . 83
4.2.1 The Cooja Network Simulator . . . 84
4.2.1.1 Required Adjustments . . . 84
4.2.2 The Directed Graph Radio Medium . . . 84
4.2.2.1 Required Adjustments . . . 85
4.2.3 The MSPSim Simulator . . . 85
4.2.4 The RealSim Plugin . . . 85
4.2.4.1 The File Interface . . . 86
4.2.4.2 The RealSim File Format . . . 86
4.2.4.3 The Live Interface . . . 87
4.2.5 The SpringLayout Plugin . . . 88
4.3 Verification . . . 89
4.4 Discussion: Limitations and Outlook . . . 89

5 DryRun: Testing Configurations 91
5.1 Test Generator . . . 91
5.1.1 Provided Functionality . . . 93
5.1.2 Running Simulations in Parallel . . . 94
5.1.3 Changes to Cooja . . . 94
5.2 CoojaTrace . . . 94
5.2.1 Wisebed Client . . . 95
5.3 Data Extractor . . . 96

6 Evaluation 97
6.1 Execution of Experiments . . . 99
6.1.1 Simulation . . . 99
6.1.2 Testbed . . . 100
6.2 Data Analysis . . . 100
6.3 Analysis of a Trace . . . 100
6.3.1 Packet Reception Ratio . . . 101
6.3.2 RSSI and LQI . . . 104
6.3.3 Discussion . . . 106
6.3.4 Conclusion . . . 107
6.4 Experiment Setup and Environment . . . 107
6.4.1 Software revisions . . . 107
6.4.2 Testbed . . . 108
6.4.2.1 Erlangen . . . 108
6.4.2.2 Brunswick . . . 109
6.4.3 Application . . . 109
6.5 Campaign: Clear Channel Assessment (CCA) Threshold Versus Radio Duty Cycling (RDC) Protocol . . . 110
6.5.1 Experiment Setup . . . 110
6.5.2 Results . . . 110
6.5.3 Discussion . . . 114
6.6 Campaign: Packet Rate Versus Channel Check Rate . . . 114
6.6.1 Experiment Setup . . . 115

6.6.2 Results . . . 115
6.6.3 Capture Effect . . . 116
6.6.4 Discussion . . . 118
6.7 Campaign: Software Versions . . . 118
6.7.1 Experiment Setup . . . 118
6.7.2 Results . . . 118
6.7.3 Discussion . . . 124
6.8 Conclusion . . . 125

7 Summary and Outlook **127**
7.1 Contributions . . . 128
7.2 Limitations and Future Work . . . 128
7.2.1 Conceptual Limitations . . . 128
7.2.2 Simulator . . . 129
7.2.3 Optimization . . . 130
7.2.4 Miscellaneous . . . 130

Bibliography **131**

Acronyms **145**

Zusammenfassung

Einleitung und Motivation Der Begriff der drahtlosen Sensornetzwerke (Wireless Sensor Network (WSN)) wurde im Jahr 2002, unter anderem von Akyildiz et. al. in der Veröffentlichung "Wireless Sensor Networks: A Survey" [Aky+02] geprägt. Sie beschreiben dort drahtlose Sensornetzwerke als Sensoren, die über Funk kommunizieren und sich dank der Möglichkeiten eines Mikrocontrollers selbständig organisieren können. Dies bedeutet, dass sie selbständig innerhalb der Funkreichweite befindliche, benachbarte Knoten erkennen und über diese Routen zum Zielkonten finden können.

Drahtlose Sensornetzwerke spielen ihre Vorteile insbesondere dann aus, wenn die Anbindung über Kabel problematisch ist. Die Knoten werden daher in der Regel mit Batterien betrieben. Die in den Batterien verfügbare Energie ist die zentrale Ressource, der sich alles andere unterordnen muss. Folglich werden energiesparende Mikrocontroller mit entsprechend geringer Rechenleistung verwendet. Die verwendeten, energiesparenden Prozessoren verfügen auf Grund ihrer Architektur nur über begrenzten Speicher und die Anbindung von externem Speicher ist meist mit einem deutlich erhöhten Energiebedarf verbunden. Dies trifft auch auf die drahtlose Funkkommunikation zu. Kommunikationsprotokolle und -hardware müssen nach ihrem Energieverbrauch ausgewählt werden.

Trotz der energiesparenden Komponenten würde ein typischer, mit zwei AA-Batterien betriebener WSN-Knoten ohne weitere Maßnahmen nur ein bis zwei Wochen laufen. Um die Laufzeit zu verlängern, muss die verwendete Hardware, insbesondere der Mikrocontroller und der Funkchip, so häufig wie möglich in den Schlafmodus versetzt werden. Im Schlafmodus sinkt der Energieverbrauch meist auf unter 1/1000.

Hierbei gerät man zwangsläufig in einen Zielkonflikt zwischen Laufzeit der Knoten und Anforderungen wie benötigte Rechenleistung, Datenrate, Reaktionszeit und Zuverlässigkeit. Häufig müssen diese Anforderungen gegeneinander abgewogen werden: Ist es besser die Daten auf dem Knoten vorzuverarbeiten und dafür weniger Daten zu übertragen (in network processing) oder mehr Daten zur Senke zu senden und die Berechnung außerhalb des WSN durchzuführen. Diese Entscheidung muss für jede Anwendung und jedes Netzwerk neu getroffen werden. Sind alle Knoten nah an der Senke, ist es bei dem obigen Beispiel wahrscheinlich sinnvoll, die Daten zu dieser zu schicken und dort zu verarbeiten. Handelt es sich jedoch um ein großes Netzwerk, bei dem die Daten von mehreren Knoten weitergeleitet werden müssen, ist es wahrscheinlich vorteilhaft, die Datenmenge zu reduzieren.

Das Finden einer optimalen Konfiguration ist häufig sehr schwierig. Um eine zuverlässige Aussage über das Verhalten des Netzwerks zu machen, müssen die Eigenschaften des Zielnetzwerks genau bekannt sein. Auf Grund der volatilen Eigenschaften der Funkkommunikation ist es aber praktisch unmöglich, diese Eigenschaften ohne Messungen in der Zielumgebung vorherzusagen.

Erschwert wird das Finden einer optimalen Konfiguration durch die knappen Ressourcen. Es ist nicht möglich das Netzwerk eine Weile laufen zu lassen, die gewonnenen Daten zu analysieren und das Verhalten des Netzwerks anhand dieser zu optimieren. Dies scheitert sowohl an der verfügbaren Rechenleistung als auch dem verfügbaren Speicher um die erhobenen Daten abzulegen. Hinzu kommt, dass das Untersuchen des Systems auf Grund der geringen Rechenressourcen das zeitliche Verhalten der Knoten stark beeinflussen kann (Probe Effect). Dies kann leicht dazu führen, dass die von den verwendeten Kommunikationsprotokollen Echtzeitanforderungen nicht mehr erfüllt werden können.

Erschwerend kommt hinzu, dass viele WSN-Betriebssysteme eine Vielzahl an Konfigurationsoptionen bereit stellen um das System optimal anzupassen. Diese sind jedoch häufig schlecht dokumentiert und es gibt zum Teil starke Interaktionen zwischen den Konfigurationsoptionen verschiedener Softwareschichten. Das Erstellen einer geeigneten Konfiguration ist daher meist nur durch einen Domänenexperten für die entsprechende Schicht möglich. Ein Anwendungsentwickler, der mit den verwendeten Softwareschichten des Betriebssystems nicht so gut vertraut ist, kann daher nur nach Versuch und Irrtum vorgehen.

Das Finden einer geeigneten Konfiguration über Versuch und Irrtum ist bei WSN auf Grund ihrer geringen Ressourcen äußert aufwendig. Viele Konfigurationsoptionen können nicht zur Laufzeit verändert werden. In einem solchen Fall muss eine neue Firmware eingespielt werden. Außer in einem Testbett, in welchem meist über eine serielle Schnittstelle auf die Knoten zugegriffen werden kann, muss dies drahtlos geschehen. Das Übertragen der Daten zu den Knoten ist zeit- und energieaufwendig und riskant. Ist die

gewählte Konfiguration in dem zu untersuchenden Netzwerk nicht lauffähig, so kann es leicht sein, dass einzelne oder alle Knoten nicht mehr erreichbar sind. In einem solchen Fall ist dann ein physikalischer Zugriff auf die Knoten vonnöten. Alternativ gibt es Techniken, die einen solchen Zustand erkennen und die Knoten in einen definierten, funktionierenden Zustand zurückversetzen. Jedoch ist auch dies mit einem zusätzlichen Ressourcenbedarf verbunden.

Erschwert wird das Vorgehen nach Versuch und Irrtum dadurch, dass drahtlose Kommunikation hochgradig nicht deterministisch ist. Insbesondere der Versatz der zyklisch ausgeführten Aufgaben zwischen den Knoten kann einen starken Einfluss auf das Ergebnis haben. Angenommen die Sensorknoten senden ihre Daten alle 60 Sekunden zur Senke. Passiert dies gleichzeitig, kann dies leicht zu einer Überlastung des Netzwerks führen. Werden die Daten jedoch gleichmäßig über eine Minute verteilt verschickt, kann dies vom Netzwerk leicht verarbeitet werden. Hinzu kommt, dass die Uhren der Knoten nicht 100 % synchron laufen, sondern immer ein wenig driften. Um belastbare Ergebnisse zu bekommen müssen die Experimente daher mehrmals wiederholt werden, idealerweise mit unterschiedlich versetzten Startzeiten.

Forschungsfrage Eine Alternative zu aufwendigen Versuchen mit echter Hardware ist die Simulation von WSN. Die Simulation erlaubt es das Verhalten des WSN zu beobachten, aufzuzeichnen und auszuwerten, ohne dass dies das Ergebnis beeinflusst. Auch können Simulationen mit unterschiedlichen Parametern leicht parallelisiert werden. Die Anzahl der möglichen Simulationen ist nur durch die zur Verfügung stehende Rechenleistung beschränkt, wodurch ein Vorgehen nach Versuch und Irrtum ermöglicht wird.

Hieraus ergibt sich die zentrale Fragestellung dieser Arbeit: “Ist es mit der Hilfe von Simulation möglich ein spezifisches WSN zu optimieren?”

Wie bereits dargelegt, muss die Software auf das spezifische Netzwerk hin optimiert werden. Dieses Vorgehen ist daher nur zielführend, wenn das zu untersuchende Netzwerk vom Simulator ausreichend genau abgebildet wird. Hierbei müssen die drei Hauptkomponenten, die ein WSN ausmachen, Software, Hardware und Funkkommunikation, betrachtet werden.

Analyse In dieser Arbeit wird daher zunächst untersucht wie typische Software für WSNs angepasst werden kann und ob der mögliche Parameterraum tatsächlich so groß ist, dass er ohne Simulation nicht untersucht werden kann. Der 6LoWPAN (Internet Protocol version 6 over Low power Wireless Personal Area Network) Netzwerkstack von Contiki hat 24 boolesche und 52 numerische Konfigurationsoptionen. Auch wenn viele von diesen Abhängigkeiten haben, so zeigt es doch, dass es nicht möglich ist alle Konfigurationsmöglichkeiten zu testen.

In einem nächsten Schritt wird untersucht, welche Aspekte bei der Durchführung von Experimenten mit WSNs beachtet werden müssen. Es werden die verschiedenen Techniken besprochen, mit denen Daten erhoben werden können, und wie stark diese das Verhalten der Knoten beeinflussen.

Einer der zentralen Aspekte bei der Verwendung von Simulation zur Bestimmung von Konfigurationsparametern für drahtlose Sensornetzwerke ist der Realismus der Simulation. Hierzu werden die verschiedenen Komponenten von WSN-Knoten untersucht und welche Umwelteinflüsse die Knoten beeinflussen. Es zeigt sich, dass die eigentliche Hardware sehr akkurat simuliert und zum Großteil auch emuliert werden kann. Die Emulation der CPU (Central Processing Unit) erlaubt es in der Simulation denselben Binärcode zu verwenden, wie auf der echten Hardware. So ist es möglich das Verhalten, inklusive Programmierfehler im Quellcode und verwendeten Werkzeugen, abzubilden. Problematisch bleiben jedoch insbesondere Programmierfehler und fehlende Funktionalität des Simulators. So wurde während der Arbeit beispielsweise festgestellt, dass die Erkennung der Kanalbelegung nicht richtig implementiert war. Dass in der Simulation ein zusätzlicher Erkennungsmechanismus fehlt, konnte jedoch nur erkannt werden, indem die Simulationsergebnisse mit denen eines echten Netzwerks verglichen wurden. Eine Verifikation von Simulationsergebnissen unter Verwendung echter Hardware ist daher unabdingbar.

Auf Grund ihrer Komplexität kann die Funkkommunikation, im Gegensatz zur Hardware, praktisch nicht akkurat abgebildet werden. Bereits kleine Änderungen in der Position und Ausrichtung der Knoten können die Verbindungseigenschaften stark verändern. Die größere Herausforderung bei der Simulation von WSN ist jedoch nicht die Modellierung der Funkkommunikation, sondern die Anpassung der Parameter, damit diese ein spezifisches Netzwerk widerspiegeln. Hier hat sich gezeigt, dass ein verhältnismäßig einfaches, jedoch gut konfigurierten Model, bessere Ergebnisse liefert, als ein komplexeres, jedoch nicht so gut konfiguriertes Modell. Um die Konfigurationsparameter zu bestimmen, muss das Netzwerk jedoch zuvor vermessen werden.

Bei der Vermessung eines WSN muss beachtet werden, dass sich die Verbindungseigenschaften über die Zeit stark ändern können. Dies ist insbesondere in einer Büroumgebung, in der sich auch die untersuchten Netzwerke befanden, der Fall: Verbindungen, in es über einen längeren Zeitraum keinen Paketverlust gab,

brachen innerhalb kurzer Zeit zusammen. Aber auch kontinuierliche Veränderung der Verbindungsqualität über Stunden konnte beobachtet werden.

Lösungsansatz Um mit Hilfe von Simulation die Software eines WSN auf ein Zielnetzwerk hin zu optimieren, wurden zwei Werkzeugsammlungen entwickelt, RealSim und DryRun. Erstere stellt Werkzeuge bereit um ein ausgebrachtes Netzwerk im Simulator abzubilden. Anschließend kann mit DryRun automatisiert im Simulator ein Parameterraum abgesucht und die Ergebnisse für die Analyse aufbereitet werden.

RealSim besteht aus zwei Hauptwerkzeugen. Das erste läuft auf den Sensorknoten und vermisst die Kommunikationsverbindungen zwischen den Knoten über einen gewissen Zeitraum. Die gesammelten Daten schickt es entweder zu einer Senke oder gibt sie direkt über die serielle Schnittstelle aus. Das zweite Werkzeug ist eine Erweiterung des Cooja WSN-Simulators. Diese spielt die zuvor gesammelten Eigenschaften des Netzwerks im Simlator ab.

Die DryRun Werkzeugsammlung unterstützt den Entwickler bei der Durchführung von Experimenten. Mit Hilfe eines Generators können Experiment-Kampanien erstellt werden. Als Eingabe bekommt der Generator das auszuführende Szenario, sowie die Konfigurationsoptionen, die ausprobiert werden sollen. Anhand dieser wir für jedes Experiment ein Skript erstellt, das die Umgebung initialisiert, die Firmware baut, die Simulation durchführt und die Ergebnisse einsammelt. Da die Skripte voneinander unabhängig laufen, können diese leicht parallelisiert und auf verschiedenen Rechnern gleichzeitig ausgeführt werden. Weitere Werkzeuge unterstützen die Aufbereitung und Auswertung der erhobenen Daten.

Evaluation Anstatt die eigentliche These zu evaluieren, dass es mit Hilfe von Simulation möglich ist eine bessere Konfiguration zu finden, wurde die zentrale Anforderung für diese These untersucht: Wie nah ist die Simulation an dem echten Netzwerk dran. Die Motivation hinter diesem Vorgehen ist, dass die Simulation zu ungenau ist um eine quantitative Aussage zu machen. Es ist daher nur möglich qualitative Aussagen zu treffen: Ist die eine Konfiguration besser oder schlechter als die andere. Wird mit Hilfe der Simulation eine optimale Konfiguration gefunden und diese anschließend auf einem Testbett evaluiert, kann lediglich gezeigt werden, dass die gefundene Konfiguration besser ist, nicht jedoch die optimale. Hierzu muss von der Simulation unabhängig die optimale Konfiguration des Testbetts gefunden werden. Die hierzu nötigen Daten erlauben eine weitergehende Aussage über die Qualität der Simulation.

Um Simulation und Testbett vergleichen zu können, wurde der Experiment-Generator so erweitert, dass er das Experiment nicht nur im Simulator, sondern auch auf einem mit Wisebed verwalteten WSN-Testbett durchführen kann. Dies erlaubte es, die gleichen Experimente mit den gleichen Parametern und somit Firmware sowohl auf dem Testbett, als auch dem simulierten Testbett durchzuführen.

Da die Netzwerkschicht die meisten Konfigurationsoptionen bereitstellt, lag der Fokus der Evaluation auf dieser. Es wurden verschiedene von dem WSN-Betriebssystem Contiki bereitgestellte Netzwerkprotokolle hinsichtlich Energieverbrauch und Paketverlust untersucht, sowie zentrale Parameter verändert. Es wurde aber auch untersucht inwieweit die Anwendungsschicht, welche die Messdaten in unterschiedlichen Raten sendet, und die Netzwerkschicht interagieren.

Durchgeführt wurde die Evaluation auf zwei verschiedenen Testbetten, die beide in einer Büroumgebung aufgebaut waren. Büros sind eine für WSN unfreundliche Umgebung. Umherlaufende Personen, sich öffnende und schließende Türen, aber insbesondere WLAN (Wireless Local Area Network), das die gleiche Frequenz wie die Sensorknoten verwendet, sorgen für starke Schwankungen in der Verbindungsqualität. Als Basis für die Simulation wurden 24 h des Testbetts aufgezeichnet. Die meisten Experimente liefen 20 Minuten. Jede Konfiguration wurde im Testbett 5 oder 10 mal wiederholt und 50 mal mit unterschiedlicher Initialisierung des Zufallsgenerators simuliert. Auch wurden für die Wiederholungen unterschiedliche Ausschnitte aus der zuvor gemachten Aufnahme des Testbetts verwendet.

Die Ergebnisse der Evaluation zeigen, dass es zwischen den Messwerten von Simulation und Testbett zum Teil deutlich Abweichungen gibt. Es gab jedoch auch Parameter mit nur geringen Abweichungen. Eine der Hauptursachen war sicherlich die Volatilität des Netzwerks. Da dieses zunächst vermessen wurde und anschließend die Experimente durchgeführt wurden, war es sehr wahrscheinlich, dass sich die Verbindungseigenschaften in der Zwischenzeit geändert hatten. Als weitere Ursache war ein Fehler im Simulator ausgemacht und nachgewiesen worden.

Trotz dieser Abweichungen war es klar möglich den Einfluss der untersuchten Parameter auf das echte Netzwerk auch in der Simulation wieder zu erkennen. Je nach Parameter war ein gut erkennbarer Unterschied der Ergebnisse der Testbetten sichtbar, welcher sich auch in den Simulationsergebnissen widerspiegelte. Dies zeigt, dass es nötig ist die Parameter auf das spezifische Netzwerk abzustimmen.

Schlussfolgerung Die Evaluation hat gezeigt, dass die Simulation von WSNs zum aktuellen Zeitpunkt noch zu ungenau ist, um als alleinige Basis zur Optimierung verwendet zu werden. Sie hat aber auch

gezeigt, dass vorgeschlagene Ansatz, das spezifische Netzwerk im Simulator abzubilden, die Optimierung eines WSN auf ein spezifisches Netzwerk aber sehr gut unterstützen kann: Es wird aufgezeigt ob und wie Parameter das WSN beeinflussen und ob es Interaktionen zwischen diesen gibt. Je nach Parameter können die Simulationsergebnisse den Suchraum entweder stark einschränken oder direkt übernommen werden.

Die entwickelten Konzepte und Software-Infrastruktur bieten einen guten Ausgangspunkt um ein besseres Verständnis über das Verhalten von WSN zu erlangen, als auch WSN-Simulatoren weiter zu verbessern. Bereits mit der Behebung der gefundenen, aber noch nicht behobenen, Mängel kann mit einer weiteren Verbesserung der Simulationsergebnisse gerechnet werden.

1 Introduction

A few years before the year 2000 multiple companies introduced new families of micro controllers. Most notably the AVR-family (e.g. [Atm11]) by Atmel and the MSP430-family [msp] by Texas Instruments, which are still very popular to this day. These micro controllers integrated memory and plenty of peripheral units. This allowed building experimental control circuits without the need for an electrical engineering background. Sensors and actors could be connected directly to the micro controller. Many tasks like low-pass filtering or PID-Control, was now possible in software – without the need of additional electrical components. Connecting a radio chip to these micro controllers, provided the basis for a new research domain: Wireless Sensor Networks (WSNs)

In 2002, when WSNs were still new, Akyildiz et. al. coined the term WSN in their paper "Wireless sensor networks: a survey" [Aky+02]: WSN are built of battery-powered nodes that communicate wirelessly to monitor phenomena. Each node must therefore have sensors to sense the phenomena and some means to communicate with other nodes, typically a radio. Akyildiz et al. distinguish WSN nodes from sensors with a simple radio interface insofar as WSN nodes have a programmable micro controller. This provides them with two main features:

- They are self-organizing. Thus, there is no need to predefine how the data collected from the sensor is exchanged and forwarded to the sink.
- They are able to preprocess the data. Meaning that they can aggregate the data or only forward relevant information.

Due to the self-organizing nature of the network, it is not required to carefully plan the deployment. Instead, the nodes can be placed as suited and they will then build up their network infrastructure autonomously. For instance, for an air-monitoring application the nodes can just be placed in every room without the need to test whether the radio is able to directly send data to a central node. Instead, it is sufficient that the node can reach at least one other node to become part of the network. Due to the self-organization, it does not matter which node that is; the network will make sure that the information reaches its destination.

As energy is limited by the batteries, low power radios with limited bandwidth are used and it is tried to keep communication to a minimum. In large networks the amount of data collected by the sensors can easily overload the available bandwidth. Fortunately, many use cases do not require all data or only derived information. Wildfire monitoring is a good example to illustrate these principles. In this use case it is sufficient to know whether there is a fire, not the actual temperature or smoke-sensor readings. It therefore is a reasonable approach that a node reports once in while that is still functioning and otherwise is quiet. Only if it actually detects a fire, it forwards this information, yet not the sensor reading, but rather some index how sure it is that there really is a fire. Also, once a fire detection application actually detects a fire, energy resources are of minor importance, but fast and reliable transmission is. Thus nodes can abandon all energy saving techniques and transmit at full power and bandwidth. If there was a false alarm, this will however require replacing the batteries.

Another example for the preprocessing capabilities is seismic monitoring. The nodes do not need to report that there is no earthquake. When the nodes do sense one, they can record the sensor readings at a

Figure 1.1: TMote Sky WSN node

high resolution and send them to a central sink at a rate that does not overload the network. In-network processing even allows nodes to compare the data collected by neighboring nodes with their own and further reduce the traffic by only sending deltas. Once all recordings have been collected at the sink, they can be analyzed.

Since Akyildiz et al. 's publication WSNs have not only been used to monitor volcanoes and forests, but also glaciers, zebras, birds, or roads, being placed on buses. [Cer+10; Beu+09; Sze+04]

Figure 1.1 depicts the TMote Sky, a WSN node that is commonly used for research applications as it is well supported by WSN Operating Systems (OSs), and other WSN tools like simulators. As of today WSNs are starting to become ubiquitous, especially in house automation. Battery driven Heating, Ventilating, and Air Conditioning (HVAC) systems that communicate wirelessly are not out of the ordinary. They are not only used for retrofitting, to save the burden of laying wires, but even in new buildings wireless solutions are often chosen over the wired option. They are often cheaper than their wired counterpart and allow for more flexibility when adjustments are required.

Although home automation and especially HVAC has reached the market, the available products are almost all proprietary island solutions that cannot interact with each other. This is likely to change in the future.[1]With the introduction of Internet Protocol version 6 (IPv6), the address space was increased from 2^{32} in IP version 4 (IPv4) to 2^{128}. While IPv4 is running out of free addresses, IPv6 now provides more than enough. This motivated the idea of a reduced IPv6 network stack called 6LoWPAN (IPv6 over Low power Wireless Personal Area Network), enabling the Internet of Things (IoT), where every device has its own IP address and can be reached from anywhere in the world. While this does not solve the problem of interoperability and it is often still not possible for products by different vendors to understand each other, 6LoWPAN is a first step in that direction.

1.1 Motivation

Despite all this progress and over a decade of research, we are still far away from "smart dust" [KKP99]. The idea was to have very cheap (10s of cent) and tiny (1 to 2 mm at each side) nodes that harvest energy using solar cells and communicate using lasers. These could then, for example, be dropped out of airplanes over enemy territory and monitor troop movements using vibration sensors or detect chemicals. Unfortunately this has never progressed beyond mock-ups as pictured in fig. 1.2. [War+02; WAP01]

In their survey Akyildiz et al. concluded:

> However, realization of sensor networks needs to satisfy the constraints introduced by factors such as fault tolerance, scalability, cost, hardware, topology change, environment and power consumption. Since these constraints are highly stringent and specific for sensor networks, new wireless ad hoc networking techniques are required. [Aky+02]

Although all of these issues have been looked into, as of today there is no integrated solution to address all of them.

Basically every solution is however limited by its energy resources. To strain them as little as possible, the micro controllers used for WSN nodes have very limited resources in terms of memory and computation

[1] According to Gartner's "Hype Cycle for Emerging Technologies, 2014" [Gar14] the IoT is currently at the "Peak of inflated expectations" and will take another 5 to 10 years until they reach the "Plateau of productivity".

Figure 1.2: Mock-up of a $16\,\mathrm{mm}^3$ solar powered smart dust mote with bi-directional communications. [War+02]

power. This however brings a dilemma: Almost any additional feature that addresses the aforementioned issue requires energy, memory and computational resources, all of which are limited. Often they must be weighed against each other: Is it better to recalculate a certain value at the costs of computational power and therefore energy, or use a look-up table which needs more memory? Is it more energy efficient to do data analysis on the node or forward the data to a sink that is connected to a power supply and therefore has unlimited power? If the computational resources are not sufficient to do the processing on the node, maybe the data can be pre-processed to reduce the amount of data that must be transmitted. To all these questions there is no universal answer, as every application and deployment has different requirements. In the end, the only solution is to tailor a system specifically to one's needs.

Yet tailoring such a system is far from easy. There are many parameters that can be tuned, especially at the network layer. While the nodes themselves are well defined and changes limited, the environment they are placed in differs and therefore also is more likely to require adjustments. Typical questions that must be answered during a set up are: How many nodes are there? Are the nodes closely packed and cause lots of interference, or are they spaced far apart and must therefore forward a lot of data to reach all nodes? What are the environmental conditions like? Are there obstacles that reduce transmission ranges or devices that cause interference? Or more generally: How reliable are the connections and how much does the connectivity change? Are there strong changes in temperature? This has an impact on the frequency generator and therefore makes the clocks of nodes run at a different pace. It also impacts the capacity of the battery.

When configuring the network one must also answer the application related questions: How much bandwidth is required? How much delay can be tolerated? Is it ok if data gets lost or must the transfer be reliable? If a reliable communication is needed, should this be achieved at the application or network layer? And if done at the application layer, how does this impact the network layer? Is there more than one application running on the node and must there be some layer that allows addressing the application and distinguishing which application sent the data? Does the application have different modes with different requirements for the network layer, for example bandwidth, timeliness and reliability.

While the answers to the application-related questions can often be answered accurately, as the application is under one's control, the network has many unknowns. The reason for this is that the actual Radio Frequency (RF) distribution is very hard to predict and low power radios are very sensitive to changes. Every surface and object reflects or dampens the signal and reflections can cancel the signal at certain spots. Moving the node a few centimeters can make a huge difference. Further on, signal strength and sensitivity of the antenna are not omni-directional. Instead the quality of the connection depends on the orientation of the node (see section 2.6.6.8, page 33).

As if this is not troublesome enough, there are many configuration options that can be used to tune the network layer. Contiki OS [DGV04], one of the popular WSN OS, has no fewer than 75 options when using the standard 6LoWPAN network stack. While the effects of some options are obvious, others are obscured or changing them has a negative impact on the performance, unless some parameter at another layer of the network is adjusted, too. Of course, these relations are seldom documented and require arcane knowledge to be tuned efficiently. Even with a good understanding of the network stack and experience, tuning the network layer can often only be done with an trial and error driven approach. Finding the

cause for the network not behaving as expected is often far from trivial. Due to the limited resources, one cannot just turn on verbose logging and then analyze the log files to find the cause of a certain behavior.

Debugging is limited in multiple ways: One of the core problems is getting hold of the debug information. The most common way of doing so is to write the information to the serial port. This approach is only suitable for a testbed, though, as it requires a wired connection – a disadvantage that WSNs try to solve. The second option is to send the data wirelessly. Yet, depending on the amount of debug information or the stability of the network, this can aggravate the issues one wants to solve. Buffering the debug-information on internal memory and sending at a later point in time, possibly with a less efficient but more reliable network configuration, fails due to the limited memory. Finally, there is the option to temporarily save the information on non-volatile memory like flash memory or SD-Card for later retrieval, either over the network or manually by collecting the SD-card, which is not very satisfying either.

Even when not using the radio and writing to the serial or some non-volatile memory, the process of debugging influences the system. The network layer imposes real-time constraints on the system: Packets must be sent in certain time-slots or forwarded in time as the input buffer will otherwise overflow. Yet due to the limited processing power of the WSN nodes, generating debug-information can easily delay the system and cause deadlines to be missed.

In the end the trial and error approach is often the only remaining solution. This brings new challenges. The configuration must be distributed to the nodes. Often it is not possible to change the options at runtime and a firmware update is required. The firmware image must first be disseminated to the nodes wirelessly, which can be a challenge in itself. And even if this is under control, one can easily disconnect a node from the network, effectively bricking it, using a not working configuration. After all, the network layer provides the greatest opportunities for optimization and therefore de-optimization.

A solution to all these issues is simulation. As it has full control over the whole WSN, it is possible to debug and therefore understand the system. Even executing instructions step-by-step does not introduce missed deadlines, because the simulated world is delayed, too. There also is no risk of bricking nodes by trying different, possibly broken configurations. And even if the simulation of the network is slower than real time, simulations can be run in parallel. The number of tested configurations within a certain time frame is therefore only limited by the available processing power.

Using simulation to tune a system has a disadvantage, though. As already noted, small changes in a real WSN deployment can change the behavior of the network. Tuning a system using simulation is therefore only reasonable, if it simulates the deployment in question and not “some setup”. This can only be achieved if the deployed WSN is mapped to the simulator accurately. Finding out whether this is possible, what the limitations are and determining what must be done to further improve this approach is the goal of this work.

1.2 Objectives

The main objective of this work is to find the answer to the question: “Is it feasible to optimize a specific WSN deployment using simulation?” The solution should also meet the following requirements:

- **Generality** Optimizing only a specific aspect of the system, for example the Medium Access Control (MAC) layer, is not sufficient. Instead, every part of the system should be taken into account. This applies to the software, as well as the complete tool chain used to generate the final binary.

- **Applicability** The solution should be applicable for a normal deployment. This excludes the use of extra hardware that is either expensive, for example to make a survey of the radio spectrum, or goes against the “nature” of a WSN, for example requiring cables, as used in a testbed.

There have been multiple approaches to optimize a network using simulation, yet to the best of my knowledge there has been no holistic approach presented before. While looking into the feasibility question, which can, as this work shows, be answered with yes, I will also look into the limitations that are imposed by the hardware used in WSN and simulation models.

1.3 Approach and Overview

The simulation of WSNs can roughly be partitioned in three parts: hardware, network and sensors. For the hardware, emulation was chosen. While interpreting every binary instruction separately introduces a

huge overhead, it allows using the same binary image that is used for the real hardware. Effects caused by different code, configuration parameters or tools are therefore also reflected by the simulator.

Sensing will be ignored in this work. The rationale is that pure sensing applications are agnostic to the values they read. Other use cases are so application specific that it is necessary to specifically model the sensor for that particular environment. This is beyond the scope of this work.

The central issue for this work, however, is whether a distinct WSN can be mapped accurately enough to a simulator to gain sufficiently accurate results to tune the system. Accomplishing this requires getting an amply accurate image of the environment in question and being able to map this data to an adequately accurate model. Both must be closely matched, as even the best model is useless, if the required parameters cannot be acquired. Similarly a precise image of the environment cannot improve the results, when this information cannot be processed by the model.

To map the network to the simulator I developed a tool chain called RealSim. It samples the connectivity between the nodes of a WSN and replays it in the simulator.

The second set of tools is called DryRun. They support the creation of test-campaigns to explore a parameter space. It supports running simulations distributed to multiple computers or experiments using a testbed. The tool chain also collects and pre-processes the data generated. It does not include tools to interpret and visualize data. Most research questions are too specific to provide tools that support features that go beyond generic data processing tools.

The approach is not evaluated by testing whether it is possible to find a better configuration using simulation, but by comparing the effect of different parameters on simulation and testbed. This is achieved by exploring different parameter spaces using a testbed, as well as a simulation reflecting that specific testbed. This method yields enough data points to prove that the similarity between simulation and testbed is not a random effect, but a result of mapping the testbed to the simulation.

One of the major challenges during the software development for this work was not the implementation of the required tools themselves, but the tool-stack they depend on. To find the cause of a discrepancy between simulation and real world experiment, it was sometimes necessary to understand the working of all components in use. This includes the software running node, specifically the OS, its configuration and network stack, but also the micro controller and radio hardware as well as their modeled counterparts in the simulator. And of course, last, but not least, radio communication and its modeling.

The broadness is also reflected in this work, as I try to give at least a brief introduction into topics that cannot be considered common knowledge, yet might be familiar to a domain expert. Also, as there is very little related work on this holistic approach and many specific publication ignore aspects that are not in their focus, yet do influence their results. I therefore try to highlight these cross-cutting concerns through this work and put the different aspects into an overall context.

1.4 Structure

The structure of this thesis is as follows:

Chapter 2: Background and Context (page 9)
This chapter will give a background and context information. I will first go into details about WSN nodes and their components. After a quick peek into typical network protocols and usage schemes, the different types of WSN simulators and their typical design are presented. The chapter concludes with an overview of the models used for the simulation of low power radio communication.

Chapter 3: Problem Analysis and Suggested Approach (page 39)
This chapter begins with a discussion of the problem of parameter optimization. This is followed by an analysis of the techniques used to adjust WSN software and how the effects of these changes can be measured. As I come to the conclusion that simulation seems the only reasonable solution for a generic approach to optimize WSN configurations, the different aspects of WSN simulators and their influence on the realism of the results are discussed. The chapter is not only theoretical, but the theories are also supported by measurements made with the tools presented in the following chapters. As the background noise is an often neglected problem, this is looked into, too, before I discuss whether it is possible to automate the optimization process. Finally, the approach of trace-based mapping of WSNs to enable simulation-supported optimization is presented.

Chapter 4: RealSim: Mapping Deployments to a Simulator (page 79)
RealSim is the first set of tools developed as part of this work. They allow tracing a real network

and replaying the collected data using a simulator. The modes of operation of the different tools are explained and how they interact with the radio hardware and simulator.

Chapter 5: DryRun: Testing Configurations (page 91)
The second set of tools developed in the context of this thesis is called DryRun. They support the creation of experiment campaigns that can be executed using simulation or the testbed. To show the power of simulation to collect information about the simulated hardware and software the tool CoojaTrace is presented.

Chapter 6: Evaluation (page 97)
To verify the approach and the tools, multiple experiment campaigns were run. For each campaign a parameter space was created, targeting different aspects of a WSN system. Utilizing DryRun the parameter space was evaluated running experiments using a RealSim-augmented simulation and a testbed. The similarity of the yielded results clearly show the feasibility of the approach. Most of the variability could be traced back to issues with the model used by the instrumented simulator.

Chapter 7: Summary and Outlook (page 127)
I conclude this theses with a summary and an outlook.

1.5 Definitions and Clarifications

The following terms are used differently depending on the context or are fuzzy. To avoid having to paraphrase these to ensure the correct meaning and simplify the reading of the text I define few terms that are essential for this work.

1.5.1 Native and Target Platform

WSN nodes are not powerful enough to run a compiler. Instead their code is compiled on a standard computer and the resulting binary code is transferred to the node. The system that generates the binary code is the native platform. The system that runs that binary code is the target platform.

1.5.2 Application

An application is a piece of software that runs on the nodes and allows the network to provide a certain functionality. A fire detection application, for example, may read temperature and smoke detection sensors, and send an alarm to the sink in case a fire is detected. Each node can run multiple applications. The application is independent of the implementation. Thus a function or program can implement multiple applications and an application can be implemented using multiple programs. Also sub-functions of an application can be placed on different nodes.

1.5.3 Deployment and Testbed

In this work I will not use the term deployment as superset of a testbed. The term deployment will be used for the situation where a WSN is used in its target environment: The nodes can only be interfaced using their wireless interface. Consequently, the nodes can only be reprogrammed using Over The Air (OTA) self-programming or after gaining temporary physical access by other means, for example using a laptop computer.

Opposed to a deployment, a testbed has additional permanently installed means of interfacing the node. Typically, this will be a programming interface to reliably reprogram the node and a serial interface to collect debug-information. Whether the alternative interface is provided using cables or a second set of WSN nodes that operate using a separate network is irrelevant.

1.5.4 Demarcation: Low-Power Wireless Sensor Networks

Classically the term WSN is used for a deployment of battery-driven nodes, which therefore need to save energy and have scarce resources. This is however changing. With smart phones and single-board computers like the Raspberry Pi[2] providing cheap and powerful alternatives, these are coming into the focus of WSN researchers and WSN are interpreted in a broader sense[3]. In this thesis, though, the focus

[2] `http://www.raspberrypi.org/`

will solely be on the classical low power WSNs as defined by Akyildiz et al. [Aky+02] and introduced in the introduction.

The research domain that is closest to and also overlaps with WSNs are Mobile Ad hoc NETworks (MANETs). While MANETs sometimes share hardware, their focus is on moving nodes. As nodes enter and leave the communication range of the other nodes, the network topology is very volatile. Compared to MANETs WSNs are more static and the volatility of the topology is mainly introduced by environmental changes. Also, for MANETs energy does not play such a superior role. A typical application for MANETs is Car-2-Car communication, where the communication partners change often.

Another domain that shares concepts, though seldom the hardware, are Body Area Network (BAN). These are networks where the nodes are distributed over the body and communicate wirelessly. They can either be implanted, glued to the skin or integrated into clothing. A typical use case is medical monitoring.

1.5.5 Node Versus Mote

The term Mote for WSN nodes was coined by UC Berkeley, which pioneered in the development of WSN nodes and called them Mote [Cul+02]. Examples are the very popular Sky and Mica Motes. In this work I will however stick to the term node.

[3]This can be seen in the focus of the accepted papers at top WSN conferences like the SenSys and IPSN, as well as the European Conference on Wireless Sensor Networks (EWSN), opening its scope to more generic sensing applications in its call for papers 2015 [CFP].

2
Background and Context

Often research is very narrow band and looks into one detail. This work however combines multiple sub-research domains and views them in the context of Wireless Sensor Networks (WSNs). These are inter alia configuration management, optimization and simulation-models, specifically those used in low-power radios. The motivation of this chapter is to support the reader who is not familiar with all of those research domains to better asses this work and the considerations made in the next chapters. Therefore, this is not a comprehensive overview, but rather a brief and biased peek into those topics.

This chapter is outlined as follows: In section 2.1 an introduction into WSNs is given. I will present some typical use cases to get a feeling for the environment they are used in and then discuss the components of a typical sensor node. Section 2.2, page 16 will present current challenges that are common to most WSN deployments, as those are the ones that can be addressed using the presented approach. As many of the issues in WSNs are network related, I will also give an introduction into network protocols, yet with a focus on their behavior towards the hardware and the radio transmission, as well as their reconfigurability (section 2.3, page 19). An overview of the types of simulators used for WSN research will be presented in section 2.4, page 20 and an insight into their architecture in section 2.5, page 23. Finally I discuss the different models used for low power radios in section 2.6, page 24. This section will give a broader overview than necessary for the understanding of this work, but is supposed to also highlight the limitations of simulating radio communication.

2.1 Wireless Sensor Networks

WSNs have already been introduced in the introduction, yet from a very technical view. In summary they are micro controllers with a low power radio, some sensors and a battery. Although not explicitly noted in the name, the nodes can of course also be used as actors, for example to switch appliances on and off.

WSNs use cases can typically be found in the following domains:

Research: WSNs have been applied to several monitoring research projects. Popular examples are ducks, zebras (see fig. 2.1), bats, woods, volcanoes and glaciers [Beu+09; Dre+16]. In these deployments the nodes have either been placed on or around the animals, respectively in the environment to be monitored. Animal-related research questions either target social behavior and or the impact of environmental changes on the behavior of animals. The researchers of environmental deployments were, for example, interested in changes of temperature, humidity and infrared radiation. In case of the volcanoes, seismic data was gathered and analyzed.

For many of the examples found in literature the use case can be considered an excuse for WSN research. Often they failed, as the goals were too ambitious. Part of the trouble certainly was that WSN are mainly researched by the Computer Science domain, yet many of the issues that came up during the deployment had their root in electrical or mechanical effects. [Beu+09]

Industry: In industry cables are still the first choice when it comes to connecting sensors. Especially in an industrial environment Radio Frequency (RF) emissions, for example caused by motors, are

Figure 2.1: A zebra with a ZebraNet collar [Zha+04]. The collar was assembled of a $16\,\mathrm{bit}$ MSP430 micro controller, a GPS receiver, a radio with a range of 1 to $5\,\mathrm{km}$ and solar cells to recharge the battery. Photo: Courtesy of Pei Zhang

Figure 2.2: HomeMatic wireless radiator thermostat. The unit combines an actor for the radiator valve and a temperature sensor. Using a wireless connection target valve position or temperature can be set and the current temperature queried. Photo: Courtesy of eQ-3 AG

not uncommon. For control tasks with real-time constraints such interruptions are not tolerable. Using radio communication instead of wires also introduces new external threats. The production process can be disrupted using a directional antenna to jam the radio communication. As even short disruptions can cause severe damage, the attacker can be hard to find. Other scenarios are espionage or directly interfering with the production process. Although it did not attack wirelessly, the Stuxnet worm [Wik15b] is an example that tempering with the production process is a realistic attack scenario.

Nonetheless WSNs can have advantages in an industrial environment. Temporary installations, for example to assess the energy usage of a production line, can profit from not having to lay cables. Also when movement is involved and/or there is little space, laying cables is often complicated. In such situations it is often easier to stick a wireless sensor to the target position. When harvesting the energy provided by the movement, the sensors can run without additional source of energy. If the sensors are placed on workpiece carriers or on the workpiece itself, which is moved around the production facility, cables are not an option either. Yet for these use cases Near Field Communication (NFC) is often used, as it is more suitable in this situation due to its very controlled reception area.

Home automation / HVAC (Heating, Ventilating, and Air Conditioning): The area of home automation, specifically HVAC is probably one of the most visible use cases for WSNs (e.g. fig. 2.2).

Figure 2.3: TMote Sky sensor node with MSP430 micro controller (8 MHz, 40 KiB flash, 10 KiB RAM) and CC2420 IEEE 802.15.4 compatible radio

Opposed to the industrial sector, where reliability is typically very important, the home automation sector is very cost sensitive. Upgrading existing installations using a WSN can be done with minimal effort. In contrast, especially for aesthetic reasons, the effort of laying new cables is often much higher.

A central WSN based HVAC system for a flat can be installed within an hour. User control units with temperature sensors are mounted to the wall, thermostats replaced by electrical control valves, and a central unit can be interfaced using a web-interface. Due to the self-organizing nature, the devices are able to set up the network autonomously. Solely the mapping of sensors and actors to a room must still be done manually.

Military: While the initial WSN research was motivated by military usage of so called smart dust [KKP99], its visibility in the classic academic research community is rather low. Besides semi-civil applications like gun-shot detection, border surveillance and chemical sensing, there is also research on topics like "smart" tank mines, troop monitoring and missile monitoring. [Dur+12]

Medical: Wireless sensors can significantly improve the quality of life of patients that require constant monitoring. Instead of having to be plugged into a socket, the sensor data is processed by a unit carried by the patient and can trigger an alarm wirelessly in case of an emergency. These medical applications are often somewhat of a corner case of the WSN domain. If the nodes communicating are placed in, on and around the body, this is rather a use case for the specialized Body Area Networks (BANs). If there is only one monitoring node, which communicates with the outside world using different base stations as the patient moves around, this rather fits in the Mobile Ad hoc NETworks (MANETs) domain. Yet as already noted in the introduction (see section 1.5.4, page 6), these research fields often overlap.

2.1.1 Sensor Nodes

Figure 2.3 depicts the common TMote Sky sensor node. Being a typical node used for research it can directly be plugged into an USB port. It provides the node with power and a serial interface to the micro controller using a USB-to-serial converter. The serial is also used to program the node. Integrated into the node is a battery holder for two AA-type batteries.

In the following I will give a brief introduction on the core components of typical academic WSN nodes.

2.1.1.1 Micro Controller

The micro controller is a chip that integrates a Central Processing Unit (CPU), memory and peripheral units. Typical micro controllers used in WSNs have an 8 or 16 bit architecture. In research the main reason for this choice is the energy consumption, for industrial applications the price plays a more important role. Nonetheless 32 bit controllers, are starting to intrude this domain. Although they use more energy per instruction, they often need fewer instructions to provide the same functionality. By applying a *race to sleep* approach, where the node wakes up, does its calculations as fast as possible, and then goes back to the energy saving sleep mode, these architectures catch up energy wise, while providing more resources, especially in terms of memory. [Ko+12]

Table 2.1: Typical energy consumption of the ATmega128L and MSP430F1611 at 3 V and 20 °C in active mode at 4 MHz using the internal clock and sleep modes. The MSP430F1611 has a sleep mode that goes down 0.2 µA. Yet in this mode it is not able to wake up autonomously using a timer, but needs an external component to wake it up. This can be the radio chip or a special real time clock chip. Numbers are based on the data sheets. [msp; Atm11]

Controller	MSP430F1611	ATmega128L
Active (4 MHz)	2000 µA	5000 µA
Minimal sleep mode	75 µA	2250 µA
Maximal sleep mode	2.6 µA	5 µA

Due to the design, the memory address range is typically limited to 16 bit. Therefore, based on the architecture, Harvard or von-Neumann, the available non-volatile memory, typically re-programmable flash memory, ranges from 2 to 128 KiB[1]. To support more complex functionality like 6LoWPAN (Internet Protocol version 6 over Low power Wireless Personal Area Network) about 40 KiB are required.

Of course a micro controller also provides means of interacting with the environment. Besides digital I/O (Input / Output) pins, they typically come with multiple ADC (Analogue Digital Converter) inputs and hardware support for data exchange (e.g. SPI (Serial Peripheral Interface), I^2C (Inter-Integrated Circuit), UART (Universal Asynchronous Receiver Transmitter)).

Table 2.1 lists the power consumption of the ATmega128L and the MSP430F1611, which are used for the popular WSN nodes MicaZ and TMote Sky. It lists the energy usage for the active state and the minimal and maximal sleep mode. More interesting than the actual numbers, is the ratio of the active state to the sleep modes. Depending on the micro controller this is a factor of ≈1000. This shows how important it is for a long running system to spend as much time as possible in deep sleep.

There are some things that must be kept in mind when interpreting these numbers, though. Going to sleep and waking up is not instantaneous. To achieve these ratios, subsystems of the micro controller must be shut down and reinitialized after wakeup. For example the clock source can be switched to a lower frequency to reduce energy consumption. When switching back to the high frequency, the clock needs some time to stabilize. Trying to keep the system as simple as possible to reduce the required time spent in an active state is not always beneficial. A complex timer system that also requires more calculations can save energy, if this allows the micro controller to wake up less often.

2.1.1.2 Radio

The choice of radio chips is mainly limited by their energy requirements. Besides proprietary solutions, IEEE 802.15.4 compatible chips are a popular solution. IEEE 802.15.4 is not only the basis of the Zigbee protocol, but also 6LoWPAN. Especially the latter makes these radios interesting for the Internet of Things (IoT), where every device, from light bulb to washing machine, can communicate via the Internet. Recently especially Bluetooth Low Energy (BLE) has become an interesting option. [Sie+12]

WSNs are typically operated using the 2.4 GHz ISM (Industrial, Scientific and Medical) band, as this can be used worldwide without an extra license. Due to the interference with WLAN, Bluetooth (BT), and microwave ovens, 433 MHz and 902 MHz have also become popular. For physical reasons these frequencies provide less bandwidth, but allow for longer ranges as the signal is not dampened as strong by obstacles. They also have the disadvantage that, depending on the region, only one of them can be used without a special license.

The large amount of interference in the 2.4 GHz band does not only hamper reliable transmissions, but also makes it very difficult to reproduce results in an urban environment: The amount of interference depends on the number of devices, the radio channel they use and their usage scheme. [LCL07] Without completely disabling noise sources, it is almost impossible to control them, as any mobile phone might connect to a WLAN access point and download an update in the background.

Although some sensor nodes use simple transceivers, most modern designs use radios that allow offloading part of the lower network protocol layers from the micro controller. Ensuring that the channel is free before sending, symbol or frame synchronization and encryption, are typical generic functions are provided by the radio chip More protocol specific functions include adding and verifying checksums, as well basic packet filtering. Offloading these functions allows the radio chip to reject a packet if the checksum could not be verified. This does not only remove the need to transmit the packet to the micro controller before analyzing the date, but can be done more efficiently on specialized hardware.

[1] 128 KiB can be addressed using 16 bit by addressing the program memory of the Harvard architecture word-wise.

Table 2.2: Energy consumption CC2420 and CC1101. Numbers are based on the data sheets. [Tex04; Tex13]

Controller	CC2420	CC1101
TX (0 dB)	17.4 mA	≈16 mA
TX (−25 dB)	8.5 mA	
RX	18.8 mA	≈16 mA
Idle	0.426 mA	1.7 mA
Power down	0.020 mA	0.0002 mA

The radio is also one of the major energy consumers. Table 2.2 shows selected energy properties of the two commonly used radio chips CC2420 and CC1101. The energy consumption has multiple interesting aspects. First of all, receiving requires approximately the same or even more energy than sending. Secondly, putting the radio into idle or even power down, which takes longer to wake up, can save a lot of energy. For example the CC1101 needs 80 000 times less energy in power down mode than in RX-mode. Network protocols that allow saving energy by temporarily shutting the radio chip down are called Radio Duty Cycling (RDC) protocols. To allow communication, they turn the radio back on at predictable points in time.

2.1.1.3 Sensors and Actors

Sensors and actors are typically very application specific. Buttons are commonly used as user-interface. Light, temperature and humidity sensors can often be found on academic nodes as these can be used in any environment. There is a broad range of analog and digital sensors. While digital sensors require a communication interface and therefore often have a higher complexity in terms of software needed for interfacing, they are easier to use. Analog sensors often need additional circuitry to ensure high quality readings. Other sensors measure proximity, vibration, gases (oxygen, carbon dioxide) and water, but also special-use-case sensors to gather seismographic or ECG data.

Light Emitting Diodes (LEDs) are the only commonly used actors, most often used for debugging and feedback. Besides switches or rather relays and motors, most actors are even more application specific than sensors.

In terms of energy neither sensors nor actors may be neglected. A simple analog sensor with a voltage divider always draws some current. If an amplifier is needed, because a signal is too weak, the drawn current can easily have an impact on the lifetime of the sensor nodes. Similarly LEDs, are often underestimated. Requiring 2 to 60 mW, a standard LED can easily become a top energy consumer.

2.1.1.4 Other Components

Besides the core components, there are additional parts that are required to run the node. While these components and their concrete selection is often of minor importance for of the overall design of the node, they do influence the energy consumption and features that can be implemented.

Power Supply While the node is powered off batteries, there are multiple ways of connecting it to the rest of the system, which can have a significant impact on the overall power consumption. Generally speaking there are three main techniques to connect the battery as power supply. The simplest is connecting the battery directly, maybe adding a diode to protect against reverse polarity. Many components can handle a wide input voltage range. This must be taken into account during the selection process.

When directly connecting the batteries to the circuitry, the input voltage will drop while the batteries are drained. Depending on the component the actual voltage can impact on the power consumption. LEDs, for example, will drain more current at a higher voltage. At a higher voltage the energy consumption will therefore be disproportionately higher. This applies to many electrical components like resistors or most micro controller.

If the supplied voltage is out of the range of the components, the easiest solution is a voltage regulator. Yet these only regulate the voltage, but have no impact on the current. They therefore drain more energy than they provide and the delta must be turned into heat, and is therefore lost.

The alternative are so called switching regulators. Very simplified they load a capacitor until it reaches a certain voltage threshold and then let the circuit run off that capacitor until it drops below a voltage-threshold to recharge it again. Especially if there is a large voltage drop, these regulators are much more effective at the cost of requiring more components. Yet switching regulators can even “pump” the voltage to

a higher level, allowing running the circuit even if the battery voltage is below the components' operating limits. Using this technique, it is possible to drain the battery further and increase the lifetime of a node.

Not only the consumption of the system must be taken into account when estimating the lifetime of a node, but also the batteries. While these have a nominal capacity, the actual energy that can be extracted highly depends on the usage scheme and environmental influences, especially the temperature. For standard batteries only the self-discharge is of real relevance, as those are designed for much higher currents than drawn by standard WSN nodes. Specialized batteries that have an especially high energy-density can however be very sensitive to the current drawn or environmental influences, causing a reduced lifetime, voltage drop or both.

External Memory As already noted, memory is one of the scarce resources. Many nodes therefore have external memory to store additional data. These memories are often around 1 MiB. Besides being able to store sensory data, they are often used to buffer firmware images for Over The Air (OTA) programming before reprogramming the node using a boot-loader.

Some micro controllers support connecting external memory to the internal memory interface. The external memory can then be addressed just like the internal memory, normally with a small penalty in respect to the timing. Often external memory is connected using a serial bus to reduce the required pin count. Accessing external memory via a serial interface is obviously not as fast as accessing it via the memory bus.

The concrete energy costs and timing of writing and reading memory depend on the memory type and the access pattern. The two common technologies for this are EEPROM (Electrically Erasable Programmable Read-Only Memory) and flash memory. EEPROM can be written byte-wise, yet is more expensive. Flash memory can also be written byte-wise, but needs to be reset, before it can be rewritten. The reset can be done at a certain granularity, typically 64, 128 or 256 B and is energy intensive. Specialized file systems have therefore been developed for this kind of memory [Tsi+09]. Both, EEPROM and flash memory, have in common that they only have a limited number of write cycles. Typically these range from 10 000 to 100 000, yet the exact number heavily depends on the selected memory.

The limited write cycles can be overcome using so called NV-RAM (Non-Volatile Random-Access Memory). A common method of implementing NV-RAM is to use the energy efficient Static Random-Access Memory (SRAM) and back it with a battery or capacitor. Lately ferroelectric and magnetoresistive RAM has become available as internal memory in micro controllers. While both techniques promise to be superior to battery oder capacitor-backed SRAM and even Flash [Tex; Eve13], it is too early to make a sound estimation about the future of these techniques.

ID Chip WSN nodes need to be unambiguously identified, to address and distinguish them. Also certain radio chips like those supporting IEEE 802.15.4 require unique addresses, which must be provided to the radio chip at startup. These unique IDs can be part of the micro controller's firmware image. This approach has the disadvantage that each node needs a separate firmware image and care must be taken that each node is programmed with the correct image. Many micro controllers provide a built-in EEPROM which is not overwritten when writing the firmware. Of course also an external memory can be used. Yet, this again requires each node to be specifically programmed with that ID, prior to the first use. An ID Chip provides a unique pre-programmed ID that can be read when starting up the node. It is therefore not necessary to specifically program the node, but any image can be programmed onto the node and can be run without further node-specific adjustments.

Real Time Clock Depending on the application running on the WSN, accurate time keeping is of varying importance. Most micro controllers support an extra clock system that is driven with a watch crystal and supports keeping track of time in low power mode, when the main clock source is powered down. This is especially important to wake up from sleep after a certain period. While more complicated, because they add yet another component that needs to be interfaced, using special real time clock chips can have an advantage. First of all they are independent of the state of the system and keep track of the time, even if the system is reset. These chips are also designed to keep time at a very low power consumption. They can be backed with a capacitor and therefore even support changing the batteries, without loosing track of the time. Another use case is during extended power-down periods. In the lowest power saving modes most micro controllers support shutting down the clock system completely. They can only be woken by an external interrupt, for example provided by a real time clock chip.

Serial Converter The most common method of interfacing a WSN node is the serial interface. Even if not deployed in a testbed, there must be some gateway to interact with the outside world. There are solutions that expose a radio chip more or less directly to a computer. However, using a standard

Figure 2.4: Shimmer3 node. Although the hardware changed it looks similar to the previous versions. Photo: Courtesy of Shimmer

WSN node and its serial as interface is often easier, as it allows reusing the already existing software infrastructure. Otherwise, it would be necessary to re-implement the network stack for that specific gateway radio to be run on the host computer.

A convenient solution is to add a USB-to-serial converter directly onto the node. That way pluging it into a computer will provide the node with power and provides a serial interface. It also allows using unused pins of the converter to trigger a reset and reliably reprogram the node.

2.1.1.5 Nodes

In the following I will feature selected nodes used in WSN research.

TMote Sky The node used in this thesis and probably the most popular node used in research is the TMote Sky and its clones the TelosB and MTM-CM5000-MSP (fig. 1.1, page 2). Its core components are a Texas Instruments MSP430F1611 micro controller and a CC2420 radio chip by the same company. The micro controller has a 16 bit RISC von Neumann architecture and runs at up to 8 MHz. Regardless of its age (it dates back to 2002), it is still considered to be one of the most energy efficient chips in its class. It provides 48 KiB of flash memory and 10 KiB of RAM. The node also has 1 MiB of external memory that can be interfaced using a serial interface. The radio chip is designed for IEEE 802.15.4 radios, and targets low energy systems. As interface to the outside world the node has three LEDs, two buttons, a digital temperature and humidity sensor, as well as two analog light sensors – one for the visible frequency range and one that includes infrared. It also has an ID-Chp with a unique 64 bit ID. It can either be powered via USB or two AA-sized batteries, which are placed in a holder that has the same size as the node.

Mica Two other typical nodes used in research are the MicaZ and Mica2, which have the same form-factor and size as the TMote Sky. [Cro04] They combine an AVR ATmega128 micro controller with a CC2420 (MicaZ) or CC1000 (Mica2) radio, flash memory, ID chip and three LEDs, but no sensors. These can be added using a special connector. The ATmega128L micro controller has a 8 bit Harvard architecture, 128 KiB program and 4 KiB data memory.

eZ430-RF2500T Another popular node for larger testbeds is the Z430-RF2500. It has a MSP430-based micro controller and a CC2520 compatible radio chip integrated in a System On a Chip (SOC). It has fewer resources than the TMote Sky (32 KiB flash, 1 KiB RAM) and no sensors, yet costs less than half as much. This makes it suitable for WSN experiments where many nodes are needed. This is often the case when focusing on the network layer.

Shimmer The first two versions of the shimmer node are very similar to the presented nodes in terms of processor and radio. Figure 2.4 shows the third version, which has a more powerful processor and a BT radio instead of IEEE 802.15.4, yet is similar to its predecessors in terms of design and interfaces. As can be seen from the casing it does not target WSN system research, but those who want to use it out of the box, especially in the context of medical applications. For sensory it has accelerometers and gyros to follow movements. Additionally many external body-related sensors are provided, for example to take an ECG, measure blood pressure or heart rate. Shimmer also provides stock firmware to ease the usage for non-computer scientists. Due to the relative high price shimmer nodes are rather seldom used in classic WSN research.

2.1.2 Network Metrics

To be able to describe a network, it is necessary to have a metric, especially for the connection attributes between two nodes. For most situations it is sufficient to know the Packet Reception Ratio (PRR) of the different links. Other factors like signal strength play an subordinate role, as it does not matter whether a packet was received using a strong or a weak signal, as long as it was received. The PRR however has a major disadvantage. It depends on many factors like the network load and interference. Also, before it is possible to acquire a PRR, it is necessary to send multiple packets.

As a fall-back metric Receive Signal Strength Indicator (RSSI) (signal strength) or Link Quality Indication (LQI) (signal quality) are often used, as they can be acquired receiving a single packet. This is based on the assumption that there is a correlation between PRR and those parameters. As will be discussed in section 2.6.5.2, page 30 this correlation strongly depends on environmental properties, especially interference.

The correlation between Signal-to-Noise Ratio (SNR) and PRR can be put into three regions: Connected, transitional and disconnected. If there is a high SNR, almost all packets arrive. If the SNR is too low or even negative, no packets arrive. In between these two regions lies the transitional region. In this region it not only hard to predict the PRR, but it is also prone to strong fluctuations [ZG03; ZK04]

To calculate the SNR not only the signal strength, but also the background noise must be aquired. While the signal strength is more or less stable, the background noise undergoes strong fluctuations. In section 3.7, page 63 I will have a closer look at the background noise I was able to survey in our local testbed.

To better estimate the quality of a received signal, the IEEE 802.15.4 standard requires the radio chip to provide a Link Quality Indication (LQI). It is supposed to provide a better metric to estimate the link quality than the signal strength. Unfortunately the LQI is not specified besides that a higher number is supposed to indicate better quality. It is therefore a vendor-specific magic number that somehow correlates to the link quality. However, for some tested radio chips it does correlate quite well with the PRR. [LC14]

2.2 Challenges for WSNs

During the introduction of the components of a WSN node I already hinted at some of the hardware-related challenges for WSNs. This work however targets optimization of software. Many of these challenges are addressed in software, yet root in the selected hardware components. Besides the radio-related issues, which are central for this work, I will therefore limit my discussions regarding hardware to those effects that influence the software. I will go a little more in depth on some of those hardware properties in section 3.6, page 52, when explaining if and how they can be simulated.

In terms of software there are many domain-specific challenges in WSN. Medium Access Control (MAC) and higher level network and routing protocols still have potential for optimization. Even standardized protocols like 6LoWPAN have interoperability problems. [Ko+11] Other research domains are efficient in-network processing, localization, and more efficient Operating Systems (OSs).

The root of almost all challenges in WSN are the limited resources. Ideally a WSN node is tiny, costs close to nothing, has sufficient memory and computation power, has a powerful radio and runs forever. A node's lifetime is the most critical requirement, as typical use cases do not allow changing the battery every other week or even month. To allow a lifetime of several months all other requirements must be subordinate to the available energy[2]. This applies to computation power, memory, radio, and any external component like sensors or actors (LEDs). For this reason typical WSN sensor nodes have 8 or 16 bit Reduced Instruction Set Computer (RISC) CPUs. Memory is typically limited by the 16 bit address bus and the cost for SRAM, typically 48 to 128 KiB flash and 10 to 64 KiB RAM. Cheaper forms of RAM like DDR use too much energy.

Similarly the radio chip must be selected by its energy consumption. This severely limits the selection of technologies that can be used. For example, Wi-Fi, although very convenient, can only be used for gateway nodes provided with a powerful battery. Due to the energy constraints, the data bandwidth is

[2]Some strain can be taken from the energy budget using energy harvesting. Nonetheless, energy harvesting is limited by its energy source or the size and complexity of the harvesting device. Wind and solar energy are not always available and must be buffered. Not only must the buffer be sufficiently large to maintain service while the primary source is unavailable, but also the energy source must be powerful enough to power the node and recharge the buffer while it provides energy. A water turbine can provide a steady source of energy, but requires a river and some effort to be installed. Vibration as source is primarily used in an industrial environment. In this case the harvester can also be tuned to the available frequencies. If hot or cold sources are available the difference in temperature with the environment can be harvested. This however is requires some effort, and of course the source itself, which ideally should be somewhat steady. [SK11; GHR13]

typically limited to about 250 kbit/s. Even though, additional RDC techniques must be used to further reduce the energy consumption.

The limited resources have a direct impact on the software. The amount of functionality that fits into the memory of a WSN node is limited, both, in terms of program and data memory. But also the computational power can become a severe limitation when implementing, for example, encryption-related algorithms. And, as already shown in the previous section, spending as much time as possible in sleep mode is the key to a long running WSN

Finding the optimal solution for a problem is often the main challenge for the software architecture. Due to the limited resources the standard solutions are often just not suitable or need tweaking to the specific use case. Having to decide between different solutions that are similar in concept is often not too complicated as each of them has their specific advantages and disadvantages. If the solutions differ strongly in concept, estimating the impact is often more complicated. A good example is for this is in-network preprocessing of data, one of the core features of WSN. Here the energy savings of a reduced data transmission must be weighed against the additional time spent processing the data. Yet even this is not as simple as it seems at first sight. The amount of preprocessing can often be varied, therefore the question is rather "how much preprocessing is reasonable?" and not "does it make sense at all?". The answer is influenced by multiple factors.

Obviously the energy consumption of the radio and micro controller play an important role. While estimating the additional energy usage of the micro controller can be done by doing measurements, or estimating the additional time the micro controller spends in active mode (e.g. using simulation), this is not as simple for the network. If the network is large and packets must be forwarded several times the energy savings are much higher than if it is a single hop network. If the environment is noisy, it is more likely that packets must be retransmitted and therefore the average energy costs of sending a packet are higher than in an environment with little interference. Different network protocols react differently to changes in load. Typically, a higher load introduces more collisions, requiring more retransmissions, causing even more collisions. Yet, how good a protocol can cope with this situation depends on the protocol, as well as its configuration.

Many of these challenges can be addressed by optimizing the system to the specific use case. This can be achieved by using different algorithms, by tuning the system to the environment it is used in or by redistributing of system resources to the different subsystems.

2.2.1 Analyzing and Debugging WSNs

Often finding a bug is more challenging than fixing it. Finding a bug, based on faulty behavior requires analyzing the system. Thus many "debugging"-tools are actually tools to analyze the system. For this reason debugging is often used as synonym for system analysis.

A common way to understand what is going on within a WSN node is to print information to the serial interface (also known as printf-debugging). Unfortunately the debug output does not only occupy valuable memory, but also has undesirable side effects on the run-time system. Functions like `printf()` require a considerable amount of computational power to format the output. This can cause the software to miss deadlines, or increase the load to a level where the system is not able to function correctly anymore. The effect that the system changes because it is monitored is called probe effect. While `printf()` is an extreme example of the probe effect, it applies to every kind of observation that affect the runtime behavior.

When debugging using the serial interface this situation is aggravated by the limited amount of output buffer. If the buffer is full, the function writing to the serial must block until the data has been written to the serial, causing the system to hang and miss its deadlines. Typically the serial hardware of micro controllers only has 1 to 8 B of buffer, but provides an interrupt to refill it efficiently once it is drained. The amount of memory that can be used to temporarily buffer the output, before it is copied to the hardware buffer by an interrupt handler, is limited by the scarce resources. With a typical transmission rate of 115 kbaud and therefore 12 characters/ms the buffer can easily be filled in a short time, if the debug output is too verbose. If the situation one is interested in is triggered under high load, this is even more problematic as this will often also cause more debug output.

Things get even more challenging when monitoring more than one node. In such a case the debug output of the different nodes must be put in a chronological order. Depending on the size of the output buffer and the timing this can require some guesswork. If, for example, the sending node has a full output buffer and the receiving an empty one, it is possible that the message that the packet was received is printed before the message that it was sent.

WSN nodes can also be debugged using a hardware debugger. These allow stopping the micro controller under a predefined condition and analyzing the whole system. While this is much more convenient, as tooling support allows analyzing the memory, the debugger only stops the micro controller, but not the rest of the WSN or even the radio. This therefore only helps tracing the behavior of the software on the micro controller until it interacts with the outside world again, unless this interaction has no timing requirements like timeouts.

The least invasive option is an oscilloscope or logic analyzer. By toggling one or more pins, state information can be monitored with accurate timing information and minimal impact on the running system. This is for example helpful to check whether certain code paths are used and how much time is spent for certain functions.

Oscilloscope or logic analyzer can also be used to debug communication between micro controller and its peripherals, by tapping the communication wires. Powerful oscilloscopes and many logic analyzers support decoding the data, so that one can directly read the data sent back and forth. Opposed to other methods of accessing this information, this method has no probe effect[3].

The presented methods require them to be hooked to a serial interface or special hardware (e.g. oscilloscope probes). General access to the serial interface is only readily available in a testbed. Physical access to nodes in a real deployment is often inconvenient. After all, a WSN was normally chosen, because laying wires would have been too complicated. All debug information must therefore also be sent wirelessly. An option is to send the debug information at the cost of additional traffic and a strong probe effect due to this additional traffic. An alternative is to use a second device, for example a WSN node operating on a different radio channel, a battery driven computer using WLAN or a smartphone. [Ren+11; Ert+06] A simulator-related approach that can help analyzing what went wrong after the network came into a bad state was presented by Osterlind et al. [Öst+09]. They stopped the network and collected the state of the nodes. They were able to reset this state on the real nodes as well as in a simulator. This is a complementary approach to the one presented in this work, as they only map the nodes' state and not the network's.

Yet another problem when debugging WSNs is their wireless nature. The networks characteristics are strongly influenced by the environment. Especially in an office environment it is very hard to control the people working there, doors opening and closing, as well as the electronic devices that operate on the same frequency (e.g. Bluetooth, WLAN and microwave ovens). [Str+14; Huo+09] This complicates reproducing results and can make recreating certain effects impossible. Even outdoors with no people around, changes in temperature can have a significant impact on the communication. [Boa+10a; Boa+10b] Recreating a radio environment is possible with a high technical effort, though. By, instead of transmitting the data over the air, using cables and specialized hardware a controlled environment can be created. [Sub+14] But even when following this elaborate approach, random effects caused by noise or marginal differences in the node's clocks (see section 3.6.5, page 58) can hinder the reproduction of certain scenarios.

Simulators Debugging WSN software using a simulator removes a lot of the hassle described. As the simulator controls everything, it is possible to access and set every parameter of the system. Reproducing a scenario therefore is not a challenge anymore, while creating realistic variance starts to become one. How to make simulations more realistic and its limitations will be discussed in section 3.6 (Realism in Simulation, page 52).

The full control over the system allows to halt time in the simulated world. The simulation can therefore be stopped, the state of the nodes analyzed in detail and continued without influence on the outcome. This analysis goes far deeper than possible on real nodes. It is not only possible to access the memory and some registers of the micro controller, but any part of the system. This can become handy when analyzing the radio, which is not as easy to monitor, because it often does not have a specific debug interface and can therefore only be accessed via the micro controller.

Being able to stop or delay the system without effect also introduces the possibility to trace the system, even if it is necessary to dereference multiple pointers and use complex triggers to stop for analysis. [SLK12] Simulators can also detect errors that the real hardware silently passes over. Examples are:

- On many architectures unaligned 16 bit reads can result in one of the two bytes being undefined.
- Due to the lack of a special error-code, a division by zero returns 0.
- Invalid commands are often ignored by peripheral hardware like the radio or require additional checks to be detected.

[3] Tapping the communication wires does have an electrical probe effect, by adding some noise and capacity to the wire. Under normal circumstance this has no effect on the communication whatsoever.

Simulators can be enriched by checks that detect these kind of problems and inform the user about them.

Another advantage of simulators is that they are only limited by memory and processing power, which is comparatively cheap. It is therefore possible to experiment with large networks, test different scenarios and run automated regression testes. All this without the need of having access to real hardware.

It must be noted, though that simulation comes at a cost. As will be discussed in section 2.4, page 20, WSN simulations will never be able to perfectly resemble the real world. One must always weigh accuracy against computational power. While emulating the rather weak hardware is normally acceptable, accurate simulation of the radio is often impossible. For the latter statistical models are used to reflect the complex effects that can be observed in a radio environment. Not only creating and verifying these models, but also mapping the real world to these models can prove difficult. Consequently, simulation can only be considered a tool that supports the development. Any results generated using simulation must be verified using real hardware. After all, a small change or bug in the implementation of the simulator can have a strong impact on the outcome.

2.3 Network Protocols

Network protocols are one of the major research topics in the WSN community. Interests of research are for example data dissemination [HC04] and collection [Gna+09], time synchronization [Mar+04], as well as anything required to achieve this, down to the MAC protocol, including routing, security, reliability, etc.

Many of the high level concepts are independent of the hardware insofar that they only require memory, processing power and some radio medium. The actual type of radio is of minor importance or can easily be adopted to. These basic requirements are however well understood. Lower level protocols on the other hand, specifically MAC protocols, can provide a significant increase of their performance by exploiting features by the hardware chip. This however makes it hard to compare their performance, especially as some protocols rely on certain hardware features to work. ContikiMAC [Dun11], for example, bases some of its decisions on the radio chip considering the radio channel as free or in use. Others make use of hardware-supported packet acknowledgments.

The characteristics of the radio chip also influence protocol design. Modern low power radios often require more energy for receiving data, than they do for sending (see section 2.1.1.2, page 12). RDC protocols try to turn the radio off as often as possible and only wake it up to listen for potentially incoming traffic in certain time-slots. If a radio chip uses significantly less energy for sending a packet than receiving, it can be advantageous to design a RDC protocol to spend more time trying to send a packet, but listen less often.

In the context of this work network protocols are yet another software component that can be configured. They are, however, of special interest due to tree aspects.

- They play an important role in terms of Quality of Service (QoS) and power consumption.
- They provide many configuration options.
- They closely interact with the radio model, one of the most critical aspects of WSN simulation.

Due to the complexity and diversity of the topic I will forgo a real introduction into network protocols. Instead, I will discuss the problems faced when trying to tune and bench mark WSN protocols, as these are the aspects I will look at in this work. A comprehensive introduction into WSN protocols is for example given in [KW07].

Tuning and Bench Marking Network Protocols Tuning and comparing network protocols is a big challenge. Often they must by tuned to a specific environment or only perform well under certain circumstances. Routing protocols, for example, often have some kind of routing table to look up the next node to forward a packet to. If the route is not known it must first be enquired by asking the neighboring nodes using a discovery protocol. If a routing table it too small, these enquiries must happen more often.

Table 2.3 illustrates this for the situation where data is sent to 4 hosts in a round-robin fashion and the routing table has only three entries. Note that even if the next node is the destination, a packet cannot be forwarded, unless there is an entry in the table. If the destination is not known, the next hop is discovered and added to the top of the table, removing the oldest entry at the bottom. In table 2.3a there is no entry for node 1 and it must therefore be found using a discovery mechanism. Once looked up, it is added to the table (table 2.3b), but the entry for node 2, which is needed next, is removed. In consequence the next hop must be looked up for each packet, providing a performance as if there was no routing table at all.

Table 2.3: Round robin routing table. When looking up nodes 1, 2, 3 and 4 in a round robing fashion, the destination required next is always pushed out of the table with three entries.

(a) Destination: 1

Dst	Next Hop
4	2
3	1
2	2

(b) Destination: 2

Dst	Next Hop
1	1
4	2
3	1

(c) Destination: 3

Dst	Next Hop
2	2
1	1
4	2

(d) Destination: 4

Dst	Next Hop
3	1
2	2
1	1

The too small routing table could be detected by calculating the ratio of forwarded packets to lookups. A very simple implementation is however likely to cause false positives and negatives, while a more complex ones require additional resources. Even if the situation can be detected, this is only one of many aspects that can cause problems. To the best of my knowledge there has been no systematic approach towards automatic detection of configuration issues at runtime.

Another example for the challenges faced when tuning a WSN protocol, is the RDC mechanism of the ContikiMAC protocol. It will also be addressed in the evaluation in section 6.5, page 110. The Clear Channel Assessment (CCA) threshold is a configurable value of the radio hardware. Based on its value and the energy level of the received radio signal the radio hardware decides whether the radio channel is clear or not. If a node wants to send a packet, it repeatedly sends it until it gets an acknowledgment from the target node or the attempt times out. All nodes turn on the radio regularly to check whether there is any data to be received. If the radio hardware considers the channel in use, the protocol assumes that another node is currently sending data. It will keep the radio turned on to receive the next iteration of the packet. If the channel is considered clear, the radio will be turned off again.

If the CCA threshold is configured too low, the channel will be considered as used, even though it is not. The protocol will interpret this as some other node trying to send a packet and keep the radio turned on unnecessarily. If the value is set too high, the radio will never be kept on to receive packets. The suitable value however depends on the environment. It is not only network specific, but even node specific and may even change over time, if environmental conditions change. The current implementation of the protocol only supports setting the threshold at compile time. Yet even if it was using an adaptive algorithm, this algorithm would have configuration options that must be adjusted to the characteristics of the network. Such an option could be the speed in which the algorithm adopts to changes of the background noise.

When comparing two network protocols it is often hard to say whether the superior protocol is really better or solely better tuned to the test environment. While this of course applies to any test setup, the situation is particularly problematic in the WSN domain, as it is almost impossible to reproduce environment conditions. Every testbed is different and even when reusing the same testbed, conditions change over time. Also, when comparing a protocol developed by someone else, there is also often a lack of knowledge how to tune that protocol properly. If wanting to select the best protocol for a certain application it is therefore often not sufficient to compare the competitors in their default configuration, but a test campaign with different configurations is required.

2.4 Simulating WSNs

Simulators try to model the real world. Depending on the requirements, the model is more or less accurate. Higher accuracy requires more computation power. Therefore, when simulating a network, accuracy must be weighed against duration. Unfortunately accurate simulation of radio communication can become very complex. Depending on the characteristics ot the material, every object reflects and/or absorbs radio waves . Even if most objects can be neglected because their impact is minimal, the perfect simulation would have to take all of them into account, which is impossible. Also, all these parameters need to be configured, which can be become very tedious. Due to this, multiple wireless network models have been developed which will be discussed in section 2.6, page 24. These problems do not apply to WSN hardware. It can be emulated or at least very accurately simulated with reasonable effort.

Most WSNs simulators can be categorized into three main categories: Network Level Simulators (NLSs) ,Instruction Level Simulators (ILSs) and System Level Simulators (SLSs). NLSs only simulate the network itself and provide an API (Application Programming Interface) to interface them (section 2.4.1, page 21). ILSs on the other hand interpret the binary code that is also executed by the real hardware (section 2.4.2). To be able to simulate a WSN, they must include some kind of network model. As their focus is on

the hardware, those network models are often not as sophisticated as those provided by NLSs. SLS are somewhere in between. Their abstraction layer is neither at the hardware nor the network, but the operating system or the OS's Hardware Abstraction Layer (HAL) (section 2.4.3). Finally there are some simulators that do not fit in those categories (section 2.4.4).

2.4.1 Network Level Simulator

NLS target simulating the network itself and the network stack. They often provide multiple radio models (see section 2.6, page 24) and a library of reference implementations of different network protocols and even network services. Due to this, it is possible to quickly implement a new protocol and evaluate it. This does imply, though that the implementation is programmed against the API of the simulator and normally cannot be reused in other systems without alterations. Also the time required to execute the code is not implicitly accounted for and must be explicitly made known to the simulator. This is normally of little impact for more powerful systems, like smart-phones or desktop computers, because the network is relatively slow compared to the rest of the system. This does not apply to the resource-constrained domain of WSN. Especially the network stack imposes real-time constraints to the system, and it is not unlikely that deadlines are missed due to the limited computation power provided by the WSN node. It might as well be that an algorithm cannot be executed at all due to memory limitations. As a consequence it is only possible to evaluate the protocol conceptually, but not its implementation on the target system.

The most popular representatives of this category are ns-2 [ns2] and OMNeT++ [omn]. Both do not originally target WSN but have extensions that are based on them. The most popular are SensorSim [PSS00] for ns-2, and Castalia [cas] and MiXiM [mix] for OMNeT++.

2.4.2 Instruction Level Simulator

The other extreme, ILS, emulate the hardware. They therefore allow running the same binary code as on real hardware. This comes at a cost: Each instruction emulated must be interpreted separately. While the hardware can do this in one tick, emulation requires the look-up of the interpretation of the command, and the execution of that interpretation. Interpreting takes much longer, because practically any instruction alters multiple registers or memory. These must be looked up and written accordingly. The simulator must also take care of peripheral units that run in parallel on the real hardware. Depending on the feature set, the simulator must make additional checks, for example whether the specific memory address is monitored for debugging purposes.

The advantage of ILSs is that they allow reusing the same code that is also flashed onto the node. They can therefore perfectly emulate a node's behavior, including issues introduced by the tool chain, for example compiler-related bugs. Also, due to the simple nature of the micro controllers used for WSN, practically all instructions have a fixed execution timing, allowing for timing accurate simulation.

Due to the comparable simplicity of the micro controllers used on WSN there are a large number of ILS available[4] – commercial and free. While they can all run binary code generated by the compiler, and therefore do not require any adjustments to the code, they do differ in their functionality, interfaces and especially in available peripherals and configurations they ship. To use these simulators in the WSN domain the ability to also emulate the radio chip, at least its digital interface, is crucial. Otherwise, it would not be possible to run the same software on real nodes as well as in the simulator. Due to this, there are currently only two relevant hardware simulators for the WSN domain: Avrora [Avr] for AVR-based systems and MSPSim [MSP] for MSP430-based nodes.

2.4.3 System Level Simulator

SLSs are somewhere in between NLSs and ILSs. They try to combine most of the advantages of both systems. They provide virtual hardware that is similar to real nodes, for example a timer subsystem, peripheral units like I/O pins to control virtual LEDs and of course a radio. However, instead of having to interface this virtual hardware using, for example a complex protocol via a bus, it can be accessed directly using function calls. Adding support for this virtual hardware to a WSN OS can therefore be done with a relatively low effort, especially if the OS already provides a Hardware Abstraction Layer (HAL).

As the WSN OS and the applications are compiled for the native platform, they run very efficiently. And, as only the HAL is adjusted, something that must be done when adding support for a new hardware

[4] Simulators for the popular ATMEL AVR controller series include [Avr; avrg; avrh; avrb; Pet; avrf; avra; avrd; avre]. [avrc] is even writen in LaTeX.

platform to the system, the impact of the changes on the OS are rather low. While a SLS is not able to simulate a specific node, it can run and test the common part of the OS and applications without adjustments, allowing uncovering programming mistakes or conceptual problems. Due to its efficiency, which is, as the code can be run natively, similar to NLSs, it is possible to simulate large networks or long time spans efficiently. As the virtual hardware is under the simulator's control, timing information can be added. For example when sending data to the radio, the virtual time can progress depending on the amount of data to be sent before returning from the function call. The concrete execution timing of the code and thus also issues like concurrency caused by interrupts are target-platform specific and can therefore not be analyzed.

The best known representative of this category is TOSSIM [Lev+03], which is specifically designed for TinyOS OS [Lev+05]. TOSSIM was developed over 10 years ago. Back then the PCs used where much weaker, while the WSN nodes used back then, like the TMote Sky, are still common today. Using ILS has therefore only become a real option in recent years.

STEAM-Sim [Mös+13] tries to not only account for the timing of the peripheral units, but also the runtime of the CPU. This is done by adding synchronization points before branches or interactions with the hardware. The time passed between these synchronization points is acquired by compiling the code for the target platform and counting the instructions in between the synchronization points and multiplying them with their execution times as done by ILSs. To improve the realism some external peripherals are even interfaced using a virtual bus. Unfortunately the source code [Ste] was published too late to be thoroughly evaluated in this work.

2.4.4 Other WSN Simulators

Cooja [Öst+07], its Graphical User Interface (GUI) being depicted in fig. 2.5, tries to combine all of these approaches. Besides simulating the network layer, it provides interfaces at the network level, at system level, which is currently only supported by Contiki, and at the instruction level. At the network level multiple, though rather simple, network models are available and can be interfaced using Java, which Cooja is written in. To interface on the system level, Contiki is compiled to the native platform and interfaced using the JNI (Java Native Interface). Finally instead of simulating the instruction level itself, Cooja integrates the Avrora and MSPSim simulators to emulate target platforms. This is a unique feature as it allows letting totally different platforms interact with each other and test their interoperability. [Eri+09]

Physical Systems The environment, specifically sensors and actors are often a somewhat ignored issue in most WSN simulators. Typically it is only possible to add some kind of stimuli file, and of course log the output. To my knowledge GISOO [Ami+13] is the only simulator that actually looked at actor-sensor feedback loops, while using a network simulator that does hardware level simulation. Using Cooja, the micro controllers' interfaces can be connected to a model in Simulink [Wik16], a tool to design, among other things, physical systems. GISOO was evaluated by modeling two water tanks that were placed at a different hight and could exchange water using a pump and a valve. Pump and valve were controlled by different WSN nodes. A third node ran the control system. The modeled system was compared to real water tanks and using real nodes.

Hybrid Simulation Besides pure simulation there are also hybrid approaches. They try to combine the advantage of simulation with the realism of a testbed. There are two main approaches to this.

One option is to use gateway nodes that have two network interfaces, a radio and the serial or USB. That way the data can be forwarded to the simulation [LRO09] or even to another testbed as supported by Wisebed [Cou+12]. Forwarding data using a second channel must however be supported by the network stack, which is not always the case, as most stacks are designed to work with a single interface, the radio.

The other option is hardware-in-the-loop simulation. This kind of simulation simulates most of the system, yet allows integrating selected hardware components into the simulation. [Ert+06; Son+12] For example when reading a sensor, the value is not generated, but retrieved using a real sensor. It is irrelevant whether the data is acquired by the same node type that is also used in the simulation or some other source. Of course, when using the same WSN node type, it is less likely that some aspect that influences the result is missed.

Similarly, the radio traffic can be transmitted using real radios. The real-time constraints of such a system depend on the abstraction layer and how much network logic is implemented on real hardware. For example, if two radio packets might cause a collision, timing is crucial. However, the timing of an acknowledgment packet is uncritical, if the timers are controlled by the simulation. It therefore does

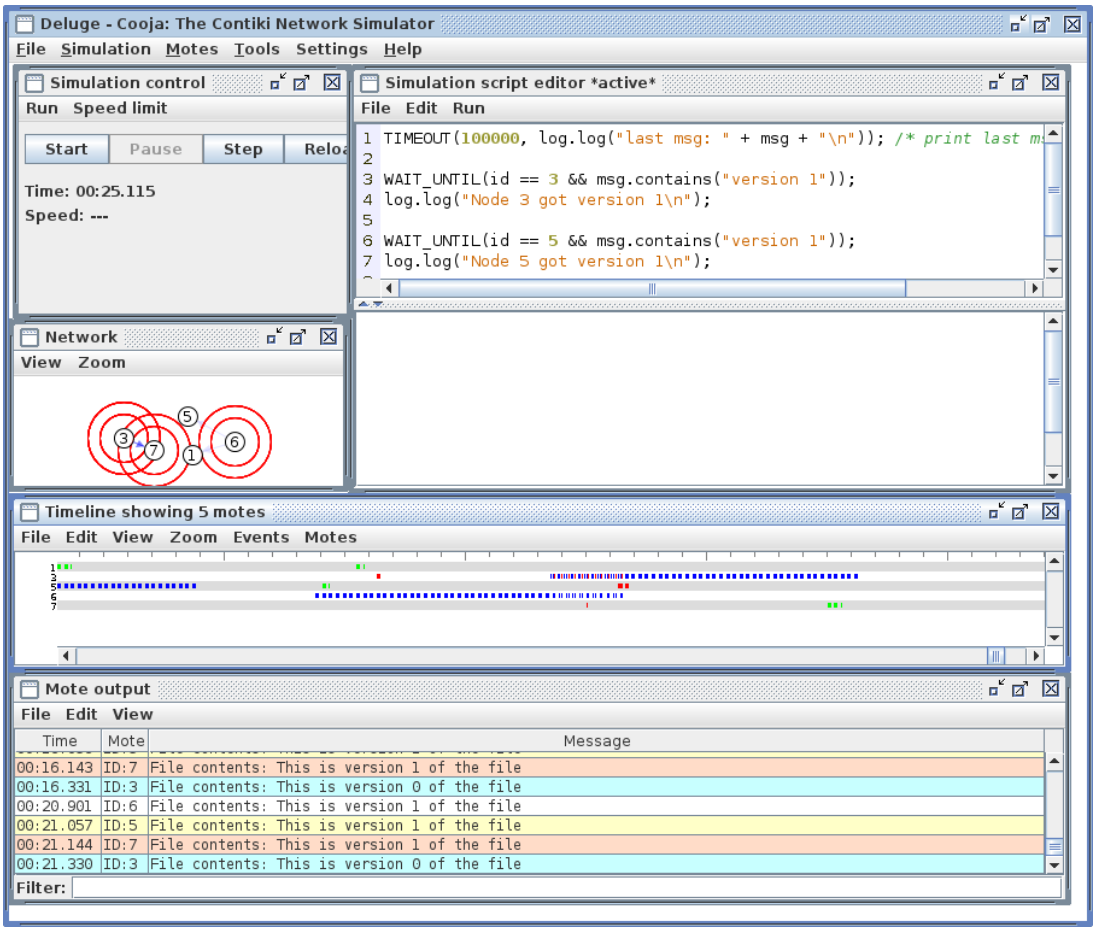


Figure 2.5: The Cooja WSN simulator. The picture shows the execution control panel in the top left. The network window visualizes the UDGM, which is discussed in section 2.6, page 24. The red rings around the nodes show that the node is sending data; the arrow from node 3 to 7 visualize a successful transmission. To the right there is a “simulation script editor”, which allows writing experiments, as well as test cases. The “timeline” visualizes the communication between the nodes. Blue represents the radio sending, green receiving and red a collision. The window below displays the output of the simulated nodes.

not matter if the acknowledgment is sent 5 s later, as long as the simulation knows that these 5 s real time must be mapped to a few ms simulation time. This assumes that the simulation has control of the acknowledgment and it is not sent automatically by the hardware.

2.5 WSN Simulator Design

Almost all simulators used for WSNs are single threaded discrete event simulators. They work off a list of events, which in turn schedule new events. Each event migrates a component of the system from one discrete state to another. To start the simulation an event to process the first instruction is queued. This event will read the program counter, execute the corresponding instruction and queue a new event to process an instruction based on the execution time of the current instruction. If the instruction interacts with other hardware components, for example by writing to a register, additional events will be queued accordingly.

Based on this scheme all components handle their events and reschedule new events either for themselves or other components they interact with. If a decision must be made, for instance whether a packet was successfully received, this is derived from a Pseudo Random Number Generator (PRNG). If initialized with the same seed, it will always give the same seemingly random numbers. This ensures that given the same input the simulation will always come to the same result. The issues caused by these seemingly random decisions will be further discussed in section 3.6.1, page 53.

All nodes of a WSN share a single event queue. This approach provides reproducibility and the lack of concurrency reduces the complexity of having to synchronize the nodes. The drawback is that modern

multi-core systems cannot be exploited and the runtime of the simulation scales more or less linearly with the number of nodes simulated. While parallelizing WSN simulations seems simple at first glance and they actually do have a great potential for speed-up, the effort of parallelizing existing simulator is probably not worth the scientific gain.

2.5.1 Hardware

As already noted in section 2.4.2, page 21, a WSN-class micro controller can be implemented with reasonable effort. This is mainly because the hardware used is relatively simple and predictable as almost all actions have constant timings. More powerful controllers have caches or pre-processing pipelines, which are often very hard to predict.

A significant amount of the complexity of a micro controller's model lies in the peripheral units. Yet to emulate a WSN node only a small subset of the peripheral features must be implemented. These are typically the serial interface, basic I/O for LEDs and buttons and a SPI to communicate with the radio. The functional complexity of the peripherals can be reduced further, by only supporting features that are actually used.

External Hardware Due to the large number of possible peripheral hardware units that can be connected to the controller, their support strongly depends on the domain the simulator is used in. While practically all of them support "connecting" a LED, more complex peripherals like radio chips or LC-displays are only implemented if there is an actual use case.

The most important and also complex hardware required for WSN simulation is the radio chip. It must provide two interfaces. One is required to communicate with the micro controller. For ILS this means re-implanting the different commands and the features behind them. For SLS and NLS a simplified interface can be created to access these features. The other interface is that to the radio model. It must reflect the characteristics of the radio chip in terms of timings, but must also interact with the radio model. This topic will be discussed in the following section 2.6.

2.6 Low Power Radio Models

Due to its complexity, the radio model of a WSN simulation allows for the largest freedom in implementation. Models reach from very simple, with very few adjustable parameters, to very complex, with many possibilities of adjustment. In the following I will give a generic overview of the different models and their properties.

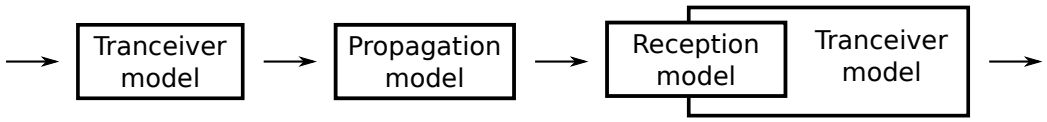


Figure 2.6: Main radio sub-models

Three main models are involved when transferring data via the radio: Transceiver, propagation and reception model. The latter is often closely integrated into the transceiver model. Figure 2.6 shows the path of data through those models.

The transceiver model provides the interface to the network simulation. When data is sent, it is therefore first passed through this model. It models the general behavior and properties of the radio chip. Depending on the chip modeled, this can include features like buffering packets and adding protocol specific information like a preamble or checksum. It might also have to provide meta-information like signal strength to the data, which is evaluated by the other models.

As the name suggests, the propagation model models the propagation of the radio signal. It includes the sending and receiving antenna, although these are often not specifically modeled, but considered isotropic (they have no directional properties). The properties a signal has when being received can either be based on the position of nodes in a virtual world or a directed graph. Properties can be the strength and/or quality of the signal at the receiver or the likelihood of the data being successfully received. The latter can well be a binary value. Most propagation models model the decreasing signal strength or a value that correlates to the likelihood of a packets being received. Based on these properties the reception model will make its decisions. Some models also support modeling the delay caused by the transmission. The short

distances and the low timing resolution of the micro controllers used in WSNs, however allow neglecting the time it takes for the radio waves to reach the receiver for most use cases.

Whether the data, and not only the signal, can be received by a node is decided by the reception model, based on the information provided by the propagation model. Aspects that can also be taken into account are other signals that are received simultaneously or the noise level at the receiver. Depending on the implementation, the received data is buffered and/or processed by the reception or transceiver model. Processing of data can include the detection of the beginning of a packet or the verification of checksums. Finally, the data is passed back to the surrounding model via the transceiver model.

In the following I make some general considerations on designing models (section 2.6.1, page 25). I will then peek into the workings of radio communication (section 2.6.2, page 25) and present some commonly used radio transceivers (section 2.6.3, page 28). After presenting some typical sources of noise (section 2.6.4, page 29), I will follow the path of the data trough the three main models that participate in the transmission of data from one node to another: Transceiver model (section 2.6.5, page 29), propagation model (section 2.6.6, page 31) and reception model (section 2.6.7, page 35).

2.6.1 Data Transmission: Bit Streams, Bytes and Packets

Radio models can work with bit streams, bytes or packets. These differ significantly in the way they are handled. Bit streams are passed to the radio model, bit by bit. If there is no buffering, a bit being sent is also received and propagated out of the radio model at the receiving node, before the next bit is passed to the sending radio. For the radio model this means that it only has knowledge about the current state and the past, yet not the potential future.

Byte streams are similar, but use a byte granularity. Here a full byte must be passed on to the transceiver before it is sent, and it must also be received completely before it can be passed on out of the transceiver model. Besides the timing, a major difference is that when using a bit stream the byte alignment gets lost. While real packet or byte based radios must be able to detect the alignment using some kind of start sequence, not having to do so simplifies the model. These models therefore assume that the start sequence is always correctly detected.

When using packets, the whole packet is passed to the radio model before it is sent via the transmission model. This is of course only possible if the transceiver has a buffer and will only start sending the packet after it received the whole packet.

Whether to use bit streams, bytes or packet based models is a cross cutting decision. Converting a packet into a byte stream and a byte stream into a bit stream is possible by splitting it up in separate bits or bytes and feeding them into the next processing unit at the appropriate time. The conversion back to a byte stream or packet can only be done when this is part of the model, for example when the transceiver buffers the received data. Otherwise the model must add a delay that does not exist in the real hardware.

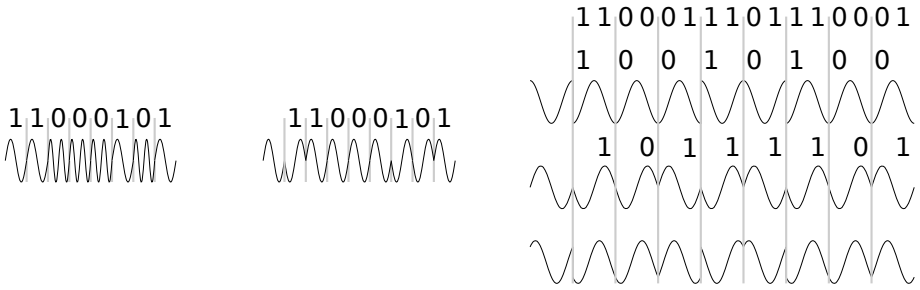
Whether to use a bit stream, byte stream or packet model depends on multiple factors. Generally the abstraction is limited by the radio that is modeled. Due to the timings, it is not possible to use bytes for a bit-stream radio. On the other hand, the complexity of the implementation decreases with the abstraction.

2.6.2 Channel, Modulation and Encoding

The channel, modulation and encoding technique used to transmit data have a strong influence on the radio properties. The most commonly supported property is the channel, as most standards dictate modulation and encoding, while supporting a selection of channels. By using different channels it is possible to run multiple independent networks in parallel. Opposed to that, the modulation and encoding scheme mainly influence reception-related properties, like the strength at which a signal can be picked up, Bit Error Rate (BER) or error detection and tolerance. Most models only reflect these properties and not their cause, especially as even the selected channel can also have a significant influence on the connection properties. [CA09]

Most WSN simulators support the use of multiple radio channels, allowing the nodes to communicate on different channels without interfering with each other. They however do not support providing channel-specific configurations and to my knowledge there are no existing extensions to add this support to the commonly used WSN simulators.

The reason for this is that although there is research in multi-channel MACs (e.g. [KSC08]), these techniques are not common enough to motivate the extension of a simulator. Either such protocols try to decrease in-network interference by communicating on multiple channels. To test these different channel characteristics is not so important. Or, they choose the channel dynamically based on channel condition.



a) FSK Modulation **b)** PSK Modulation **c)** QPSK Modulation

Figure 2.7: Examples of commonly used modulation techniques

To evaluate such algorithms it must not only be possible to configure the channels separately, but also adjust the conditions over time, which increases the complexity.

For the discussed reasons, modulation and encoding are often ignored by simulators, as it is just assumed that all nodes in a WSN use the same modulation and encoding. They do, however, influence the properties of the different models. I will therefore give a quick introduction into modulation and encoding techniques commonly used in WSN.

2.6.2.1 Modulation

Almost all radio communication techniques are based on the same basic principle: They have one or more carrier frequencies and a way to encode the payload on to these frequencies (modulation) by changing them. For example, the analog data for audio radio reception, as used in a car radio, is either transmitted used amplitude modulation (AM) or frequency modulation (FM). Amplitude modulation uses a fixed carrier frequency and modulates the signal by changing the amplitude of the signal. Frequency modulation on the other hand uses a fixed amplitude and varies the frequency based in the input signal. Using the latter is more complex, yet has the advantage that the amplitude and therefore the signal strength is not weakened, even when there is silence.

Frequency Shift Keying The FSK modulation is the digital counterpart of FM and one of the simplest modulation techniques. The sender switches between two (or more) discrete frequencies, where each frequency represents a value. Figure 2.7a illustrates this by using twice the frequency to modulate a 0 as used to for 1. In reality the two frequencies are chosen closely together to reduce the bandwidth required.

Binary Phase Shift Keying Instead of using two different frequencies it is also possible to shift the phase of the signal by 180°. There are multiple ways to encode data using PSK. One is to use one phase for 1 and one for 0. As the phase is just an offset in time, the receiver has no prior knowledge about the currently sent phase and some kind of synchronization mechanism is required to agree on the phases. An alternative is that a phase shift represents 1 and no shift 0 as shown in fig. 2.7b. This has the advantage that the receiver and transmitter do not need to synchronize their phases.

Quadrature Phase Shift Keying By using two carriers at the same frequency, but with an offset of 90° it is possible to encode twice as much data. Figure 2.7c illustrates this. The first wave shows the encoding of the odd bits, while the second wave encodes the even bits with an offset of 90°. The third wave is the result of mixing the two waves. As the data density of the three modulation techniques presented in fig. 2.7 are to scale, it can be clearly seen that QPSK requires only half as many changes as the binary PSK to transmit the same amount of data.

Interference All the presented modulation techniques have additional variants with different properties in terms of robustness, BER and complexity of the sender and receiver. Yet the examples show the following:

- The receiver can lock on to the frequency and therefore tolerate small amounts of interference.
- The base frequency allows sender and receiver to implicitly synchronize their clocks.
- If the signal is interfered, this can be detected based on the signal shifting its phase either at the wrong point in time or by the wrong amount.

How well tolerating and detecting interference works, depends on the modulation technique and the complexity of the receiver.

2.6.2.2 Encoding

The payload sent at the data layer must somehow be mapped to the two or more discrete states provided by the modulation technique. I will present three commonly used encoding techniques, the advantages of encoding and discuss how they can influence the radio model.

Non-Return-to-Zero (NRZ) Encoding NRZ is the simplest method of encoding. The radio transmits a certain state depending on the data. This technique is for example used for RS232, SPI and others. In the following the state for a 1 will be represented by a high and 0 by a low level.

NRZ is very bad in detecting misconfiguration and invalid data. If, for example, some other radio continuously sends a carrier frequency, the transceiver cannot distinguish it from a valid signal, as NRZ modulation does require regular state changes. Likewise, a misconfigured baud rate can cause a wrong interpretation of data as illustrated by fig. 2.8. When the sender is sending at half the baud rate a 1 will be read as two 1s and a 0 as two 0s as illustrated in fig. 2.8a. This cannot be detected by the receiver. Even when sending at a higher rate, the receiver does not necessarily detect this, but just outputs random data. Whether the value returned for the ? in fig. 2.8b is a 0 or a 1 or is even detected as an error, depends on how and when the data is sampled by the receiver.

a) Sender sends at half the frequency.

b) Sender sends at twice the frequency. The value returned at ? depends on the receiver.

Figure 2.8: Example of using NRZ encoding resulting in undesired results.

There are of course techniques to address this. For example, by sending a preamble which contains a certain pattern. Only after receiving this pattern, the following data is considered valid and is then also aligned to a start sequence. Notwithstanding, it is also possible to use checksums, yet they can only verify the correctness of already received data.

Manchester Encoding Some issues presented by NRZ encoding can be addressed using Manchester encoding. It is a technique developed at the university of Manchester to encode clock and data, using a single signal. It also ensures that a transmitted signal switches between the two signal states regularly. [For00]

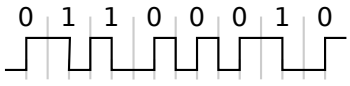


Figure 2.9: Manchester encoding

This is done by transmitting a high or low signal depending on the data to be transmitted and then toggling the signal (see fig. 2.9). As the clock is also integrated into the signal, it is possible to discard signals with the wrong baud rate as noise. A side effect is that the signal is balanced, meaning that the number of transmitted 1s are the same as 0s, which is beneficial for the High Frequency (HF) radio design. This even allows for a lower BER at the same data rate. [Tex07]

Manchester encoding also impacts the error model, as there is a high chance that a bit-flip of the encoded data can be detected: In such a case the signal is not toggled as expected. When simulating at the data layer, bit-flips therefore become less likely, while a new error class “encoding” is introduced. Whether a detected encoding error is propagated as connection loss or explicitly as encoding error depends on the hardware.

Pseudo-random Noise (PN) A technique to make the transmission more robust against interference is PN. It maps the data to a sequence that provides a certain amount of statistical randomness. The random nature makes it easier to distinguish the signal from noise: Really random signals are unlikely to have the same “random” sequence and non-random signals do not look random. For reasons that go beyond the scope of this work, PN spreads the bandwidth of the signal at the radio layer. From the RF point of

Table 2.4: Symbol to chip mapping used for the IEEE 802.15.4 PN encoding

Symbol	Chip Sequence
0	11011001110000110101001000101110
1	11101101100111000011010100100010
2	00101110110110011100001101010010
3	00100010111011011001110000110101
4	01010010000101110110110011100011
5	00110101001000101110110110011100
6	11000011010100100010111011011001
7	10011100001101010010001011101101
8	10001100100101100000011101111011
9	10111000110010010110000001110111
A	01111011100011001001011000000111
B	01110111101110001100100101100000
C	00000111011110111000110010010110
D	01100000011101111011100011001001
E	10010110000001110111101110001100
F	11001001011000000111011110111000

view, PN is therefore a method to achieve a Direct-Sequence Spread Spectrum (DSSS), which makes a signal more robust to noise by increasing the bandwidth of the signal.

IEEE 802.15.4 implements PN by dividing each byte into two nibbles, and encodes these as bit sequences, so called chips (see table 2.4). The receiver chooses the chip with the lowest hamming distance to the received signal. The chips used by IEEE 802.15.4 reach a hamming distance of at least 12, allowing the radio to work with a low SNR. [Goy+10] This shows that not only the modulation as discussed in the previous section, but also the encoding and the ability of the radio to tolerate and detect transmission errors has a significant influence on the PRR.[5] Wu et al. [Wu+12] closely analyze how noise, for example by other IEEE 802.15.4 radios, impacts the PHY layer of IEEE 802.15.4, yet the effects they observe are hard to transfer to simulation.

2.6.3 Example Radios

To give a more hands-on impression, I will briefly introduce tree representative radios, and discuss how they affect the radio model in terms of data transmission. When actually modeling these, there are a lot more things that must be considered.

Bit Stream Radio: CC1000 The CC1000 radio chip [Tex07] is a commonly used in WSN networks. It transfers the data using FSK modulation and NRZ encoding using two different frequencies – one for a 1 and one for 0. Being an unbuffered bit stream radio, it sends the state of an input pin and sets an output pin according to what is currently received. For this it has two modes: Synchronous and asynchronous. In the synchronous mode, the input is sampled at a certain rate and the state is held until the next sample. The radio provides a clock signal to the micro controller. Thus the micro controller must only ensure that the pin is written in between clock signals. In the asynchronous mode, the radio chip just sends the current state of the input pin and provides no clock signal. The same options are available at the receiver side: The chip can either sample the signal at a certain rate and provide a clock signal to synchronize on, or set the output to whatever it is receiving.

In synchronous mode the radio has hardware support for Manchester encoding. Encoding errors are propagated and it is even possible to configure the tolerance. This must be considered when modeling a reception model.

[5] Although it seems obvious that due to this encoding scheme and the additional checksum any successfully received IEEE 802.15.4 packet can be considered legit, this is not the case. As Goodspeed shows in [Goo14], it is possible to hide a valid packet within another packet without this being obvious when looking at the raw data. This situation can only be emulated when simulating at the PHY layer, though.

To not take noise for a signal, the CC1000 requires a balanced preamble (same number of 1s and 0s) to allow the receiving radio to synchronize on the sending radio. Most models just assume that this works correctly and drop the first few bits until the radio is synchronized. Once the radio is synchronized, it starts forwarding the received data to the micro controller until the synchronization is lost.

Packet Radio: CC2420 The CC2420 radio chip [Tex04] is a packet-based radio, specifically designed for IEEE 802.15.4. It has hardware support for many features of IEEE 802.15.4, which allow reducing the load on the micro controller. The features include adding and verifying checksums, CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) and addressing.

Although it is a packet radio and can buffer a whole IEEE 802.15.4 packet, it works at the byte layer. Meaning that it is not necessary to transfer the whole packet before starting to send, but only the first few bytes. The rest of the packet can be transfered when the radio is already sending. The same applies for the receiving side. It is possible for the micro controller to start reading the packet before it is received completely. When using the checksum verification, its outcome is attached to the packet's payload together with the RSSI and LQI. The RSSI and LQI will be discussed in section 2.6.5.2, page 30.

Multifunction Radio: CC1101 Finally there are multifunctional radios like the CC1101 [Tex13] that can process bit-streams, byte streams as well as packets. The main difference between those modes is that when in packet mode the radio supports additional features like checksums. The CC1101 is a very versatile chip. It supports six modulation modes, and Manchester encoding. It does however not support more complex encodings like PN.

Modeling a chip like this can become extensive. The models therefore often only support a subset of features, for instance only supporting the packet mode. It also introduces additional complexity to the other models, as different modulation and encoding techniques have different attributes as discussed in section 2.6.2, page 25.

2.6.4 External Noise Sources

Unless in a deserted area, man-made noise influences the WSN. Especially the commonly used 2.4 GHz ISM band is shared with many other protocols like WLAN and BT, but also microwave ovens and other electronic devices. [Boa+09; LCL07] These have to be explicitly added to the simulation.

External noise can be modeled in two ways. The easier solution is to feed the external noise directly into the reception model. This is a suitable approach when not knowing the exact sources, but having some knowledge about the noise at the node, for example by tracing it. The alternative is adding special transmitters that send bogus data that cannot be decoded by the WSN node's transceiver. For the Cooja simulator this has been implemented by Boano et al. [Boa+11]. Of course both methods can be combined.

If modeled as transmitter, the bogus data is distributed via the signal propagation model. As it is possible to add multiple sources, their characteristics can be modeled separately. For example BT supports frequency hopping and therefore interferes the signal at regular, yet short, intervals. WLAN on the other hand does not change channels, but is likely to use multiple at the same time. Consequently the amount of interference is different, and of course also depends on the load. Microwaves on the other hand have a very regular profile when turned on, but are normally turned off most of the time. [Zac+12]

Using separate noise sources is also important when using mobility, as is done when researching MANETs. In such a case the nodes can move in and out of the range of the different noise sources and their specific characteristics.

2.6.5 Transceiver Model

The transceiver model provides the interface from the hard and software to the wireless simulation. Based on the type of simulation the interface consists of simple API calls for NLSs, or the simulation of the hardware interface (e.g. SPI) for ILSs. The functionality of the model includes all functions that are also provided by a normal radio chip. Examples are buffering, handling of the CSMA mechanism, encryption, adding and verification of checksums, and of course sending of data. As already noted, there are different reception models, which implement more or fewer parts of the transceiver. Depending on the design and abstraction layer they can include buffering and verification of packets.

Abstraction A transceiver model can model a generic abstract or a specific existing transceiver. The generic model has a generic interface and its properties must be adjusted to meet the specific requirements. These kinds of models are typically used in generic network simulators like ns2 [ns2] or Omnet++ [omn].

A model of a specific chip tries to emulate this specific chip, including its configuration options and timings. Simulators that support instruction level simulation of specific nodes, must also implement a transceiver model that simulates a specific chip. Otherwise, the major advantage, being able to reuse the code without adjustments, would be lost, as the transceiver does not behave as expected by the software running on the node. This does not only include correct understanding of commands sent by the micro controller, but also includes state transitions and timings. Implementing a specific transceiver does, however, not necessarily mean that the interface must match that of the original chip as long as its behavior can be modeled. This is for example reasonable when running radio-chip specific experiments using a NLS.

2.6.5.1 Interacting Properties

Besides the data being sent, there are properties that can influence decisions made by other models. These properties must be exposed to the other models.

Signal strength The signal strength is a parameter that is set by the transceiver model, processed by the propagation model and evaluated by the reception model. A stronger signal can reach nodes, which are further away. Depending on the reception model it can also decide which of two concurrently received signals can successfully be received.

The signal strength of real radios depends on the output power of the radio chip and the antenna. The latter will be discussed in section 2.6.6. Adjusting the output power is supported by many radio chips and must therefore be provided to the propagation model, if changing it is supposed to be accounted for.

State The state of the transceiver must be accessible to the reception model. If the radio is not in receive mode, it can obviously not receive anything. Additionally, when in receive mode, the reception model might need to inform the transceiver model that it is currently receiving a signal, because certain chips (e.g. CC2420 and similar) exploit this information to the micro controller.

Channel, modulation and baudrate While almost all chips used in WSN support adjusting the channel, some also support to alter the modulation and baudrate. Most WSN radio models only account for the channel. This information must be provided by the sending part of the transceiver model and be evaluated by the reception model. This information must therefore be passed through the propagation model. How this is done is left to the model.

2.6.5.2 Receive Signal Strength Indicator (RSSI) and Link Quality Indication (LQI)

RSSI and LQI are two parameters that indicate the signal strength and quality and are provided by many radio chips. IEEE 802.11 (WLAN) and IEEE 802.15.4 solely define that the RSSI should have a value from -128 to 127 and that a higher number represents a stronger signal. How the actual signal strength is mapped to the RSSI, is however undefined. The commonly used CC1000 and CC2420 chips for example have a linear mapping from the input power in dBm to the RSSI value. (Chen et al. investigate the accuracy of this linearity. [CT10]) As an approximate mapping is given in the data sheet, the "RSSI" is often given in dBm. This can be considered inaccurate or even wrong, because the RSSI is only an indicator and not the signal strength. In addition the offset provided by the data sheet is only a approximation.

The signal strength however, and therefore the RSSI, is a bad indicator for the link quality. In a low noise environment a weak signal can be picked up, while in a high noise environment the same signal might not be receivable at all. In addition, the input power is even increased by the noise, resulting in a higher RSSI, yet inferior reception. For this reason the IEEE 802.15.4 specifies the LQI, which should give a signal to noise ratio estimation and ranges from 0 to 255. [IEE11]. The only further specification is that it should have at least 8 unique values. Even the CC2420 data sheet [Tex04] vaguely describes the LQI as a kind of "chip error rate" (chips are part of the IEEE 802.15.4 encoding). Without further information the transceiver can only be modeled experimentally. In consequence any software making decisions based on RSSI and LQI must be especially adjusted to the radio chip used.

2.6.5.3 Radio-Chip Features

When implementing a concrete radio chip, the transceiver-model must behave like the actual chip. Here the same considerations apply as to ILS: The features used by the radio driver must be implemented,

while those unused can be ignored. Not implementing them however introduces the danger of not noticing their lack when using them later.

In the following I will discuss some basic features provided by many radio chips. The challenge is that these are often implemented slightly differently or documented vaguely. This makes it hard to model them accurately.

CCA The CCA indicates whether the channel is assessed as clear. Some RDC protocols like ContikiMAC rely on the information to decide whether to keep the radio in reception mode. However there are multiple options to base this decision on. The CC2420, for example, supports selecting one of the following conditions

- the received energy is beneath a certain threshold
- no valid IEEE 802.15.4 data is received
- both of the above

These are the modes required by the IEEE 802.15.4 at the time the radio chip was published. The current IEEE 802.15.4 specification requires even more modes.

Especially the requirement of detecting IEEE 802.15.4 traffic is problematic because it is vague. Unfortunately at least the data sheet of the CC2420 does not go into further detail. Most likely the channel is considered in use when one or more IEEE 802.15.4 chips (see section 2.6.2.2, page 27) can be decoded. Without radio-chip specific documentation or measurements this is however guesswork.

That this is an issue is also reflected in the fact that the detection of valid IEEE 802.15.4 data is not implemented in model of the CC2420 model provided by the MSPSim ILS. The impact will be discussed in section 6.5.2, page 110.

CSMA-CA Many chips support built-in CSMA-CA. Before sending data the chip checks whether it considers the channel as clear (CCA). It can either ignore the send-command or delay sending the data until the channel becomes clear. The concrete implementation depends on the chip.

Timings Implementing correct timings is not always easy. For some operations, data sheets only set an upper bound. This can root in different causes, like the algorithm used, state changes not being executed in constant time or the operation being synchronized with chip-internal events.

In practical use this is often irrelevant. When however trying to reproduce a certain effect using simulation, this can become relevant.

An example for this is the charge-pump that loads a capacitor with a voltage that is higher than the supply voltage and needed for the HF part of the chip. After turning on the radio, the capacitor must be charged to its operational voltage. The time needed to reach the required voltage depends on the supply voltage, the capacity of the typically external capacitor and whether it is still loaded from before the radio chip was turned off.

Encryption En- and decrypting data can be very computation intensive. Providing hardware support allows weak micro controllers, as used in WSN, to encrypt their data sent over the radio faster and more efficiently. [HNL08] Many radio chips therefore support this feature.

When simulating the radio chip, the data can be encrypted or only marked as encrypted and handled accordingly. Both solutions have advantages. Implementing the encryption can be considered the safe way, especially as it can be verified by comparing the simulation results to data sniffed from the real network. Just marking it as encrypted is easier to implement and allows easy logging and analysis at the radio layer. Yet care must be taken that all checks are correctly implemented, like verifying that sender and receiver use the right keys and all necessary delays are added.

Most chips also allow encrypting data without actually sending it. The encrypted data can then be read back to the micro controller. If this feature is used, the encryption must be implemented. Otherwise it is hard to verify that the system is working correctly.

2.6.6 Signal Propagation Model

When a node sends out data, the signal gets weaker the further it has to travel. It will also be reflected by surfaces, which can cause diffraction. Depending on the position of the receiver, this can degrade or even improve the signal quality and strength. Additional effects are refraction, polarization and scattering. The sum of all these effects is reflected by the signal propagation model.

The model does not have to model the actual effects. It is sufficient to provide one or more estimated, calculated or fixed properties on which the reception model can base its decisions. Typical properties are signal strength, distance or the PRR.

In the following I will discuss the propagation models commonly used in WSN simulations.

2.6.6.1 Directed Graph Model

The directed graph is the generic approach, which all other approaches can be transferred to. For each directed connection between nodes it has set attributes. Set in this context means that they are not derived from other properties, yet can be changed at any time. This also means that each connection must be configured separately. Most other models try to derive the attributes based on the position of the nodes.

2.6.6.2 Unit Disk Graph Model

The UDG model defines one or more regions (disks) based on the distance to the sending node. The simplest form is reception/no reception: Either the node is within the disk and can receive the data, or there is no reception at all. This can be extended by adding a region where the signal cannot be received, but causes interference, as implemented by the Cooja simulator. The nodes in the interference range are not able to receive any data while they are interfered. Other extensions support setting a PRR, possibly for multiple disks. Using a function of the distance between two nodes to provide the PRR is sometimes implemented, too.

The UDG model is based on no physical effect besides the radio range being limited. It is therefore suitable for quick proof of concept experiments as it provides an unrealistic, yet easy to set up environment.

2.6.6.3 Free Space Model

The free space model applies to radio waves being transmitted in free space. It assumes that the energy transmitted is evenly distributed over the surface of the sphere growing with distance. This means that the signal strength is proportional to the inverse of the square of the distance between sender and receiver.

$$P \propto \frac{1}{d^2} \tag{2.1}$$

Consequently doubling the distance will reduce the signal strength to a quarter.

The general equation to calculate the path loss is as follows:

$$PL = 20\log_{10}(f) + 20\log_{10}(d) + k \tag{2.2}$$

It calculates the path loss (PL) in dB based on the distance d and frequency f. k is a constant which depends on the units used for d and f. This is necessary because dB is unit-less. [Wik15c; Rap02]

2.6.6.4 Two Ray Model

The two ray model extends the free space model by one reflection. This is the case when the sender and receiver have a line of sight, the ground is more or less flat and reflects the radio waves. In classical radio communication this represents two radio communication towers.

Figure 2.10 shows an exemplary graph of the power versus the distance. The graph shows multiple drops where the direct signal in eliminated by the signal reflected by the ground. The concrete properties depend on the position of the communication partners and the wave length. Due to the small distances in the WSN domain the effects of this model can be seen at low heights of about 70 cm. [Sto+09] Another WSN-related use case are Unmanned Aerial Vehicles (UAVs).[BFG14]

The pass loss for the two-ray model is calculated as follows:

$$PL = 40\log_{10}(d) - \log_{10}(Gh_t^2h_r^2) + k \quad \text{for } d > (4\pi h_t h_r)/\lambda \tag{2.3}$$

It calculates the path loss (PL) in dB based on the distance d, the height of transmitter h_t and the height of the receiver h_r. G is a constant that can be derived from the characteristics of the antenna. Details can be found in the relevant literature. k is a constant which depends on the units used by the other variables. This however only applies for distances that are beyond the drops that can be seen in fig. 2.10. The minimum distance depends on the hight of sender and receiver and the wavelength λ. [con15; Rap02]

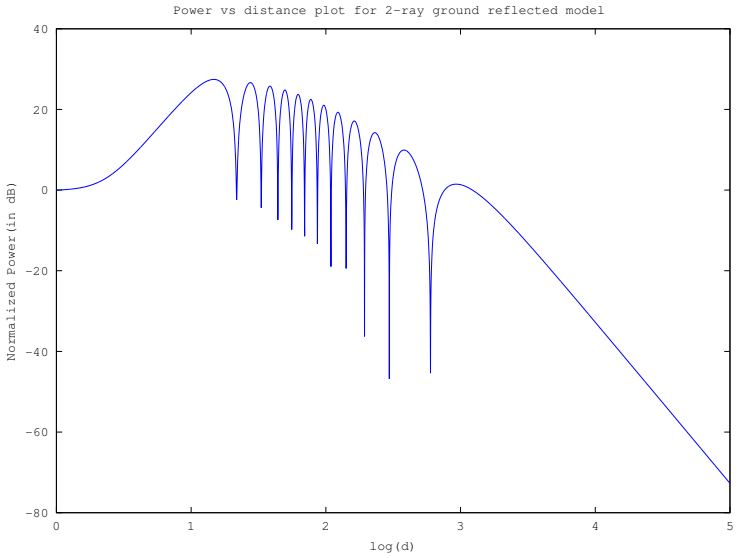


Figure 2.10: Exemplary graph of the signal power versus distance of the path-loss model. Based on work by Catverine (Own work) [CC BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0`)], via Wikimedia Commons [Cat13]

2.6.6.5 Log-Distance Path Loss Model

The log-distance path loss model is a generalization of the free space and two-ray model.

$$PL = PL_0 + 10\gamma \log_{10}(d/d_0) \tag{2.4}$$

It is based on an empirically determined reference path loss PL_0 at a distance d_0. Based on this, the path loss PL can be calculated at the distance d. The factor γ depends on the environment and must be acquired empirically. There are however reference tables that support selecting a suitable γ without the need of experiments. PL and PL_0 are given in dB. PL_0 is transceiver and frequency dependent and must therefore be provided for each combination. [Rap02]

2.6.6.6 Log-Normal Shadowing Model

The Log-distance path loss model (eq. (2.4)) will always return the same path loss for a given distance. However, as discussed before, not only the distance, but also the position relative to other objects that influence the signal are of importance. To take environmental effects like reflections into account, the log-normal shadowing model adds a random factor to the log-distance path loss model (eq. (2.4)).

$$PL = PL_0 + 10\gamma \log_{10}(d/d_0) + X_\sigma \tag{2.5}$$

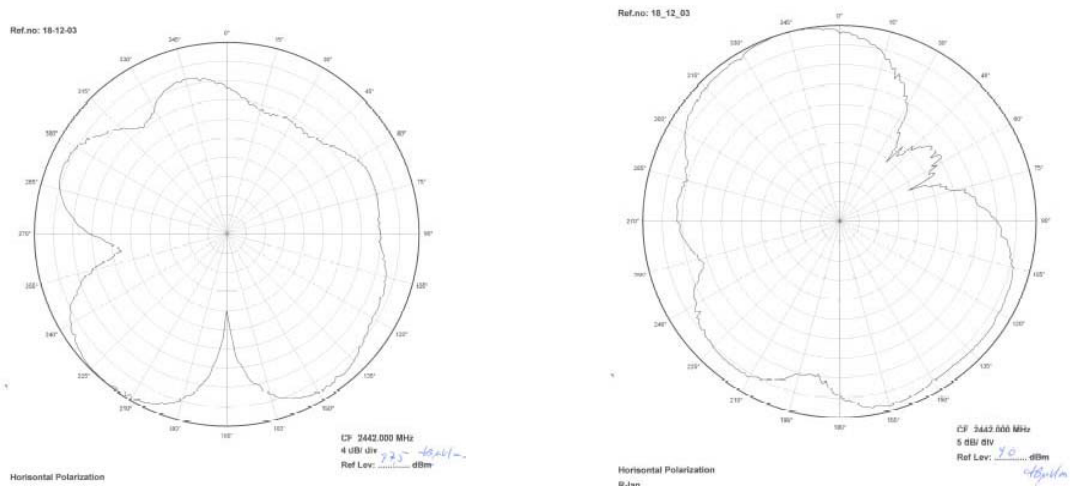
χ is a zero-mean Gaussian distributed random variable in dB with a standard deviation σ. Just as γ, σ mist be found empirically and depends on the environment, transceivers and frequency. [Rap02]

2.6.6.7 Raytracing

In the log-normal shadowing model (eq. (2.5)) effects like reflection, dampening or diffusion are represented by X_σ. Raytracing tries to model these effects using a model of the environment. This technique is commonly used for light when rendering 3D environments. In the radio model each radio sends out “rays”. If they hit a surface they get reflected, dampened and/or scattered. The signal quality can then be derived from the “rays” reaching the receiver.

2.6.6.8 Additional Effects

The presented models cover the basic mechanisms. They can be extended by modeling certain aspects. In the following I will briefly present the aspects of radio irregularity, mobility and changes over time.



a) horizontal mounting **b)** vertical mounting

Figure 2.11: Radiation pattern of the internal inverted-F antenna of the TMote Sky [Mot06]

Radio Irregularity It is not possible to build an isotropic antenna, which evenly radiates signals in all directions. The irregularity is mainly influenced by the antenna design, but also the physical environment near the antenna. Especially the design of the node influences its characteristics. Figure 2.11 shows the radiation pattern of the TMote Sky node. One can see that in the vertical mounting the signal strength from 20 to 50° is much weaker as form 90 to 0°. Thus when only considering this one axis, there is a high chance of getting a good, but also a chance of getting a bad reception when randomly placing the node.

For the log-distance path loss and log-normal shadowing models the signal strength is part of γ. Yet most models only allow setting γ without a direction specific part. Stoyanova et al. add a directional component into their two-ray model [Sto+09]. They use measurements taken from a real node which are then approximated to reduce the complexity. This radiation model is very node-specific and must be recalibrated for other nodes. To tackle this issue Zhou et al. suggest a Degree of Irregularity (DOI) model [Zho+04], which provides a more abstract description.

Mobility Mobility plays a minor role in WSN as this is rather a problem of the MANET domain. Yet nodes, and therefore simulators, are often shared between the WSN and MANET domain. There are different mobility models that model people, cars and both in different environments. [Asc+10] The positions provided by such a model can then be evaluated by the signal propagation model.

Time Although radio connections change over time, for example due to changes in temperature, or by other aspects like people walking around, this is seldom natively supported by simulators. The most common form of changes in the signal propagation model are mobility models.

Instead of defining concrete values for certain times, Castalia allows to define Probability Density Functions (pdfs) to describe, for example, the received signal strength. [Bou11] For each time step the pdf sets a likelihood that the signal has a certain value, as well as factors describing how strongly it will change in comparison to the previous value.

As part of this work I addressed this by developing the RealSim plugin, which allows adjusting Cooja's Directed Graph Radio Medium (DGRM) properties at predefined times. [rea]

2.6.6.9 Trace-Based Models

Instead of modeling the connection attributes, it is also possible to take traces from real networks and replay them in the simulator. This can be done by sending packets and calculating PRR, signal strength and quality from the received packets. These traces can then be used to configure a directed graph (section 2.6.6.1). It is a prerequisite that the data collected by the trace can be mapped to the properties provided by the directed graph.

There have been multiple approaches to improve the radio propagation model by using traces from real deployments. [HL10; Mar+10] Instead of using traces directly, Kamthe et al. use them to derive a model using a combination of multilevel Markov models and a mixture of multivariate Bernoulli distributions.

[KCC13] By modeling the environment, instead of replaying it, it is possible to generate synthetic traces that have the same characteristics as the real environment.

Using traces to configure a directed graph-based radio model is also the approach chosen for RealSim, which is presented in this work (see chapter 4, page 79).

2.6.7 Reception Model

The reception model implements if and how data is received. While there are several general concepts, the concrete behavior is strongly coupled with the transceiver model: The reception model must respect the different configuration options of the transceiver model. Examples are the radio channel, modulation type or baud rate. Additionally transceiver-specific characteristics like the handling of the capture effect, which will be discussed in section 2.6.7.3, page 35, must be taken into account.

I will first discuss how the impact of noise can be modeled to influence the reception model and then present different commonly used reception models.

2.6.7.1 Noise

The noise at a receiver has two sources: The first is the noise floor, which is influenced by the antenna, the amplification circuit and interference by other components on the node. It can even be seen if all other sources are blocked off. Antenna and amplification circuit are influenced by the temperature and the production process. They can therefore differ from node to node and also change over time, when the environmental properties change. The changes are normally small enough that the noise floor can be considered more or less stable, especially when there is external noise, which has a much stronger impact. [ZK04]

The second type of noise is caused by external sources. Implementing the sources as separate entities that transmit bogus data via the signal propagation model was already discusses in section 2.6.4, page 29. Yet external noise can also be modeled to be fed directly to the reception model. This approach is suitable if there is knowledge about the characteristics of noise at the node, for example by measuring it. Recreating virtual noise entities based on data measured from nodes is possible [Liu+10], but likely to become tedious. Implementing the external noise in the receiver model is rather easy, as it only requires to adjust the background noise and there is no need to interface any other model.

2.6.7.2 Basic Models

In the following I will discuss the basic reception models. The more complex and powerful models will be discussed afterwards.

Binary Model The binary model is often used in combination with the UDG model. A node is either within the range of another node and receives all data or none at all.

Packet Reception Ratio and Bit Error Rate Model The PRR and BER models are similar to the binary model, but use a Random Number Generator (RNG) to decide whether the packet or bit can be received. The chance of receiving the packet or bit is based on the parameters provided by the radio propagation model. It either provides PRR/BER directly, or they are derived from some other value like the signal strength.

Signal-to-Noise Ratio Model The SNR model is a variation of determining the transmission range using the propagation model. Instead of providing a likelihood of reception, the propagation model only provides the signal strength. If the signal is stronger than the background noise plus an offset, the signal can be received.

Interference Model The presented models are rather simple and therefore the handling of interference is simple, too: If two signals reach a node at the same time, neither can be received. Being able to receive a strong signal in the presence of a weak signal, the so called capture effect, is reflected by the Signal-to-Interference-Plus-Noise Ratio (SINR) model, which is discussed in the following.

2.6.7.3 Signal-to-Interference-Plus-Noise Ratio

The SINR model is the most advanced commonly used model. It does not only look at the noise floor, but also at all other interferences. The basic algorithm works as follows:

1. Pick the strongest signal

2. Add up the energy of all the other noise sources, including other signals.
3. If the ratio of the strongest signal to the sum of all the other sources of noise is above a certain threshold the signal is successfully received.

This check must be repeated for every change of the signal strength. Thus, it is possible that only half of a packet is received when another node transmitting data causes an interference resulting in a loss of the original signal.

Capture Effect The presented algorithm implies that a strong signal can be received without problems in the presence of weaker signals, as it is the case with real radios (*stronger-first*). Similarly, if the node starts receiving a stronger signal, it will lose the original signal and can start receiving the new signal (*stronger-last*). Unless using very simple radios, at least the signal loss should be detected quite reliably, especially if more complex modulation and encoding schemes are used (see section 2.6.1, page 25). [Zha+07] How fast the transceiver is able to synchronize to the new signal and can start successfully decoding it depends on the modulation technique, as well as the properties of the transceiver. [KSM08]

Without modeling the capture effect, the modeled packet loss is higher than it would be using real nodes. This results in an over-pessimistic PRR, which is especially strong in a dense environment with many nodes. The effect is well studied in the literature [SKH06; LW09; Whi+05; HL10; Zha+07].

Unfortunately there are no or contradictory statements in the literature about the stronger-last situation. For the CC2420 Kiryushin et al. [KSM08] suggest that both packets are lost, Dezfouli et al. and Son et al. [Dez+14b; SKH06] claim that this is possible and the data sheet is very vague on this [Tex04]. The quality of the experiments presented by Dezfouli et al. suggest that the stronger-last packet is successfully received, at least for the commonly used CC2420 and CC1000 radio chips.

Dezfouli et al. also suggest handling the preamble, which is used to synchronize the receiver and contains no data, differently [Dez+14a]. They define a number of settling bits as the minimum length of the preamble that must be received. If anything before that is interfered, this does not hinder the successful reception of the packet.

The SINR model only decides whether the signal can be received, not whether it is successfully received. This can be done based on a fixed PRR/BER or dynamically, as presented in the following.

2.6.7.4 SINR based Bit Error Rate and PRR

BER or PRR can be derived from the SINR (section 2.6.7.3). This is more realistic than a fixed value, as a successful reception is more likely with a large SINR, as when the signal can just be received. The formula required for this depends on the modulation technique. For a NonCoherent FSK (NCFSK) modulation, as used by the CC1000 radio, the theoretical probability of a bit error is

$$Pr(bit)_{NCFSK} = \frac{1}{2} \mathrm{e}^{-\frac{1}{2}\frac{\mathrm{B}}{\mathrm{R}}\,\mathrm{SINR}} \tag{2.6}$$

[Rap02]

where $SINR$ is the SINR not in dB and B is the noise bandwidth and R the bit rate. Thus increasing the bit rate also increases the likelihood of a bit error exponentially. The inverse applies to the SINR. However, as the SINR is given in dB, which is based on $\log_{10}$, this cannot be directly applied to the signal strength. Doubling the offset of the signal to the noise increases the likelihood of a successful reception by about 1.35 ($e^{\log_{10}(2)}$).

For the 16-Offset Quadrature PSK (OQPSK), as used by IEEE 802.15.4 the probability can be calculate as:

$$Pr(bit)_{16-OQPSK} = \frac{1}{30} \sum_{k=2}^{16} (-1)^k \frac{16}{k} \mathrm{e}^{\,20\,\mathrm{SINR}\left(\frac{1}{\mathrm{k}}-1\right)} \tag{2.7}$$

[Dez+14a]

As the presented formulas are theoretical values, they only allow an estimation. At least using a SINR based BER does not match experimental results [CT10; Pet+06; SA05; Dez+14b].

2.6.7.5 Experimental Models

The models presented are all derived from theoretical considerations. Yet there are also approaches deriving a model from experiments. They found that low power radios can be grouped in three categories: Good (PRR>90 %), transitional and bad (PRR<10 %) [ZK04; ZG03; Dez+14a]. If the SNR at a node is above a certain threshold, the data can be received perfectly. Increasing the signal strength cannot improve the

connection properties anymore. If the SNR is below a certain threshold, there is no chance of receiving the data.

Zuniga et al. [ZK04] analyze the transitional zone and show that the size of the transitional zone depends on the hardware, as well as the environment. They explicitly note that its attributes are influenced modulation, encoding, output power, frame size, noise floor and channel parameters. They also describe techniques to determine properties to model the transitional zone. These have to be acquired for each combination of output power, modulation and encoding. Based on these analyses Zuniga et al. present a model derived from the log-normal shadowing propagation model and the SINR reception model.

3
Problem Analysis and Suggested Approach

In this work I investigate whether it is possible to use simulation to optimize a specific Wireless Sensor Network (WSN) deployment. As the optimization should also be deployment specific, the network must also be reflected by the simulation. In the following I will look at different aspects of this approach and analyze how they influence the approach. The problem can be grouped in 3 thematic pillars: Optimization, running and evaluating experiments, and simulation:

Optimization

- **What are potential optimization objectives?** section 3.1, page 40
  Before it is possible to optimize a WSN, it is first necessary to know what to optimize for. Besides application specific, there are also general goals.
- **How severe is the problem?** section 3.2, page 42
  If there are only a few options, then they can easily all be tested. However, if there is a huge configuration space, it might not be possible to test all possible combinations, even using simulation.
- **How can a system be adjusted?** section 3.4, page 45
  To implement tool support, the tools must be able to adjust the parameters of the system. It is therefore important to understand how the system can be adjusted by these tools and how this impacts the result.
- **Which strategy can be used to optimize a system?** section 3.3, page 44
  Bases on the requirements and the conditions faced, a strategy to find the optimal solution must be chosen.

Running and Evaluating Experiments

- **Which metrics can be generated to compare experiments?** section 3.5.1, page 49
  To find a good configuration it is necessary to compare the outcomes of the executed experiments. Before it can be determined which configuration has the higher packet loss or lower energy consumption, these numbers must be acquired first.
- **Is is better to run experiments using simulation or a testbed?** section 3.5.2, page 52
  Results from a testbed can be considered more accurate than from the simulator for the time being. Running numerous experiments using a testbed can also become very tedious and even infeasible.
- **Are there other methods to get good configuration parameters?** section 3.8.1, page 73
  If it is possible to calculate or otherwise predict the effect of a change in configuration, maybe the overhead of simulation is not required.

Simulation

- **Is the simulation accurate enough?** section 3.6, page 52
  For the feasibility of the presented approach, this is probably the most important question. To

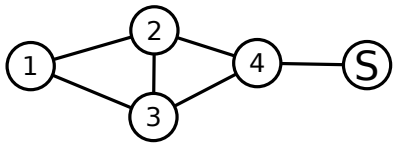


Figure 3.1: Example network to show the impact of failing nodes. All nodes send data to the sink S. If node 4 fails the whole network fails, while the impact of other nodes failing is smaller.

enable a deployment specific optimization, the simulation must be realistic enough, so that changing a configuration option has the same effect on a simulated WSN as it has on the real WSN. If the simulator is not accurate enough, mapping the real world to the simulator will not improve the simulation results.

- **How can the deployed network be mapped to the simulator?** section 3.8, page 73
  There are two main requirements that must be met to allow mapping the real world to a simulator. Firstly, it must be possible to acquire the relevant information from the network. Secondly, the simulation model must be able to handle this information.

In this chapter I will also look into background noise (section 3.7, page 63). The analysis of the realism of simulation approaches has shown that this is an often neglected topic.

3.1 Optimization Objectives

As already pointed out, the requirements for a WSN are very application specific and one must almost always weigh one optimization goal against another. Consequently there is no generic answer or solution to optimizing a WSN. In the following I will discuss some typical optimization targets and their challenges.

Lifetime The Lifetime of a network is certainly one of the main optimization goals. Increasing it by minimizing energy usage or increasing energy storage are the classical ways of increasing the lifetime.

How long a WSN however lives, depends on the use case and its requirements. The life of the network ends when it is dead. Yet when does this happen? When the first node dies? When last node dies? When the number of active nodes drops below a threshold? When the number of nodes that can reach the sink drops below a threshold? When the density of nods in a certain area of the deployment drops below a threshold? Which of these possibilities applies is use-case specific and cannot be generally answered.

Quality of Service Often the QoS is a better indicator for the lifetime of an WSN than energy: Even if all nodes have a sufficient power-supply, the network can be considered useless or dead, when it is not able to fulfill its purpose. Elongating the lifetime of a WSN network by putting the nodes into sleep mode so often that they cannot fulfill their task is unlikely to be the goal of an optimization.

Using fig. 3.1 I will illustrate that optimizing the lifetime of a node does not necessarily mean optimizing the lifetime of the network. Suppose all nodes send data to the sink S. If node 4 fails, no data can reach the sink, and the network can be considered dead. If any of the other three nodes dies, however, 3/4 of the network still work. Depending on the use case this might be sufficient.

Surmising that 3/4 of the network are sufficient to meet the QoS-requirements, this allows for new considerations: By increasing the transmission power of node 2 it might be possible to sacrifice the lifetime of node 2 to the advantage of nodes 3 and 4, and thus increase the overall lifetime of the network.

There are multiple metrics that allow to define a minimum QoS. Typical network related QoS metrics are node-to-node and source-to-destination Packet Reception Ratio (PRR), delay and jitter. Often such metrics are more complex, though. For example, it might be sufficient to receive every other measurement, resulting in a PRR of 50 %. However losing three packets in a row, while providing a PRR of 90 % might still be a problem.

Similar effects can also happen at the software layer. If a node is too busy handling network traffic, it might not have enough time to sample sensor data. Or it has too many direct neighbors whose data it is supposed to aggregate, causing it to lose data. [Vil+11] looks into similar problems in the context of placing operators of a data stream system for WSNs.

Fault Detection and Tolerance To achieve a hight QoS it is necessary to detect or even tolerate faults. Independent of the fault tolerance and fault detection mechanism, they always require additional resources. For example: To tolerate packet loss the data must be buffered at the sender until it is successfully received at the receiver. There must also be some kind of fault detection mechanism to detect whether the packet was received at its destination or not. Alternatively the packet can be sent using two independent routes. This approach allows tolerating the loss of one of the two packets. This will require additional processing and radio bandwidth resources.

As the two examples show there are different ways of tolerating a fault. They differ in the types of faults that can be detected, reliability, robustness and required resources. Most techniques have attributes that must be adjusted to the specific use case. This is for example the amount of memory that is assigned for buffering packets or the timeout before a new packet is sent.

In the WSN domain fault detection is not limited to the network. Node failing due to empty batteries, broken hardware or software bugs are not uncommon. Failing soft- or hardware is typically addressed using some kind of watchdog: If a certain event (e.g. the scheduler is called or a special packet is received) does not occur within a certain time frame the node is reset. Besides resetting the node, it is also possible to replace the firmware with a "golden image" that is known to work. [BS09; HC04]

While most network-related faults are well known from non-WSN-related network research, the constrained resources often make it impossible to use the developed techniques. The scarce resources also make it necessary to weigh the different techniques and their resource requirements against each other and tune them to the specific deployment. After all, memory resources used for buffering packets might be better utilized if there is practically no packet loss in a specific deployment. Further techniques are discussed for example in [Ulb14].

Scalability Scalability is once again a very use-case specific optimization goal. If the exact number of nodes is known beforehand, the network can be optimized for this. For example the size of the routing table can be adjusted accordingly. Yet, when adding a node at a later point in time, this may cause severe performance degradation. It is possible to address this by using routing algorithms that do not require a routing table (e.g. [AK04]).

The network cannot only scale in size, but also in density. If the nodes are placed closer together, each node is able to reach more nodes, reducing the number of hops required for a packet to reach its destination. At the same time this increases the chance of collisions. How well a protocol can cope with this, depends on its algorithm and configuration.

Most of the presented optimization goals can be addressed by adjusting the software as well as the hardware. Again, these must be weighed against each other: Is it better to optimize the software to use less energy, or add additional batteries to the node. Alternatively, adding additional or more powerful nodes, adjusting the antenna or using a different radio chip can help solve the problem. Any of these adjustments change the prevailing circumstances and almost always entail the requirement to re-optimize the software.

Breaking Down and Verifying Optimization Goals Unfortunately, there is no such configuration option as "Let the node run for `N` days". Instead there are many different aspects that all influence the lifetime of a node. Examples are the number of packets being handled, the amount of computations done, the algorithms used at the application, network and Operating System (OS) layer. Any of these are influenced by multiple configuration options.

Consequently, to address the different goals, these must be broken down into sub-goals. For example, to reach the goal of increasing the lifetime of the network, the derived goal of saving energy is addressed. A sub-goal to saving energy is letting the micro controller spend as much time in sleep-mode as possible. However, one must be very careful when accounting the time spent in sleep mode as the transition is not instantaneous and adds to the energy budget while the processor is halted. As discussed in section 2.1.1.1, page 11 it can be of advantage to use more time with computation, if this can reduce the number of wake-ups.

Similar effects can be seen with Radio Duty Cycling (RDC) protocols. If the radio spends too much time in sleep-mode, this can cause packet-loss, because the packets cannot be transmitted. Retransmitting the lost packet can require more energy, then can be saved by having the radio spend more time in sleep mode.

These are only two examples that illustrate that optimization is far from trivial and care must be taken when defining a sub-objective. It must therefore be verified that measures taken really contribute to the

Table 3.1: Contiki's configuration options for the 6LoWPAN network stack, broken down by protocol. The list does not include options to turn on debugging. The numbers are based on Contiki version `fe0a0423` from 2013-08-11.

Protocol	Boolean	Numeric
ConitikiMAC	9	19
IP	8	8
UDP	1	1
TCP	2	8
ARP	0	2
6LoWPAN	2	2
RPL	2	11
	24	51

overall goal. The many aspects that influence a requirement also make the development of optimization tools difficult, as they must be able to address different, likely contradicting interests.

3.2 Severity of the Optimization Problem

As discussed in the previous section, different configuration options help reaching an optimization goal. In fact modern WSN OS like Contiki[DGV04], RIOT-OS[Bac+13] or TinyOS[Lev+05] provide a huge amount of different configuration options, especially at the network layer. These allow tailoring the system to the requirements of the deployment and use case: Network topology, bandwidth, lifetime, responsiveness, reliability, connectivity and so on. When using Contiki's default 6LoWPAN (Internet Protocol version 6 over Low power Wireless Personal Area Network) network stack, the developer is faced with a total of 75 configuration options distributed over 7 protocol layers (table 3.1). Of the three mentioned OS ContikiOS is certainly the one with the most configuration options. This does not mean that the other systems cannot be adjusted in the same way, only that the options are not exposed as configurable option. Also this only includes those parameters that can be set using C PreProcessor macros. Additional options can be set at runtime using parameters passed to the corresponding functions.

The provided options can be divided into two categories: Those that can be decided by reasoning and those that require experiments. Many of the Boolean options fit into the first category as many enable or disable a certain feature. Whether a feature is needed, can often be decided by analyzing the application.

Also some of the numeric parameters can be decided by reasoning or at least by making an educated guess. The others can only be roughly estimated or require experimental evaluation. The number of parameters that must be evaluated experimentally depends on the application and the expertise of the person configuring the system.

Two aspects of the parameters complicate matters. Many parameters interact with each other. Thus adjusting one parameter also requires adjusting the other parameter to gain an improvement. Secondly, as stated, optimization is always a process of weighing one aspect against another. Unfortunately these are often not only two, but multiple aspects and these are not necessarily linear.

An example is the Channel Check Rate (CCR) for ContikiMAC that will also be analyzed in the evaluation (see section 6.5, page 110). It defines how often the radio wakes up to listen for packets. If the value is too low more packets are lost and must be retransmitted requiring additional energy. If it is too high energy is required to listen for potential packets that are not sent. In addition a higher CCR reduces the time a packet needs to reach its destination. In the end it is possible that a value must be chosen that is not perfect in terms of energy to meet requirements in terms of timeliness.

The interaction between parameters is often not trivial. Sometimes they are within a layer, but often also spread across multiple software layers. In the evaluation I will, for example, look at the correlation of resolution in time and therefore the number of packets sent by the application layer and the CCR, which is a configuration option at the RDC layer.

That there is no perfect default value also highlights another issue: Typically the original developer of the configuration option chooses a value that he or she thinks is suitable for most situations, or, and this is probably more common, a value that is suitable for the local use case. Due to the large number of parameters, knowing which parameters to adjust and which to leave at their default value is far from trivial.

3.2.1 Discussion: Documentation

With the analyzed software being a mainly non-profit open-source project configuration options are often added without sufficient documentation. Commercial products have the reputation to have better documentation, and to my experience these seldom provide undocumented features. However, there is also a lack of freely available commercially driven alternatives. TI, the manufacturer of the CC-radio-chip-series, for example refers to Contiki. I will therefore discuss the examples from the Contiki Code as well as from the Light Weigth Mesh protocol provided by Atmel [Atm14].

The following code snippet shows an excerpt from the central IP configuration provided by Contiki:

```
#ifndef UIP_CONF_IPV6_QUEUE_PKT
/** Do we do per neighbor queuing during address resolution (default: no) */
#define UIP_CONF_IPV6_QUEUE_PKT 0
#endif

#ifndef UIP_CONF_IPV6_CHECKS
/** Do we do IPv6 consistency checks (highly recommended, default: yes) */
#define UIP_CONF_IPV6_CHECKS
#endifg
```

Listing 3.1: Contiki OS: core/net/uipopt.h:192ff

While the effect of activating `UIP_CONF_IPV6_QUEUE_PKT` is explained, the consequences can only be judged by someone familiar with the protocol: Where are the advantages and disadvantages of having a separate package queue per neighbor? And in which scenario should it be used?

A similar problem arises with `UIP_CONF_IPV6_CHECKS`. While it says that the consistency checks are highly recommended, the question arises why this is the case. Is this a counter measure against errors that happen during transmission? If so, is not the IEEE 802.15.4 stack robust enough? Or is this based on the assumption that another system injects broken packages? Can it be therefore be deactivated, if not using other systems, or might it even address bugs in the implementation provided by Contiki? Also the impact of the feature in terms of memory and runtime overhead is unknown. This makes it almost impossible to decide such issues without experiments, or prior experience.

As already noted, testing the two options provided by Boolean options can be done with a limited effort. For numeric options this is however not as easy, especially if a reasonable range is large or cannot be estimated. Many of the numeric options are rather easy to cope with, like `UIP_CONF_MAX_CONNECTIONS`, which sets the maximum number of Transmission Control Protocol (TCP) connections a node can track. Analyzing the applications running on the node should give a clear number of required connections. Finding a suitable value for `NETSTACK_RDC_CHANNEL_CHECK_RATE`, which sets the rate at which the RDC layer checks for new data, is not as trivial. Its impact depends on the RDC protocol used, and a good understanding of the protocol is required to estimate the implications that come with the change. As I will show in the evaluation, the properties of the network and the amount of traffic being sent must be taken into account when choosing a suitable value (see section 6.6, page 114). As both attributes depend on the use case, it is not possible to have a perfect default value.

But even commercial products require an expert to be configured. The following snippet is taken from the Atmel's Light Weigth Mesh protocol.

> `NWK_DUPLICATE_REJECTION_TABLE_SIZE` – number of entries in the duplicate rejection table. This table is used to detect and reject frames that already have been received and processed by the stack. This table is used to resolve:
>
> - Loops in the network topology. Sometimes the routing algorithm may create a situation where a frame is routed in a loop between a few nodes
> - Already processed broadcast frames, received from the neighboring nodes
>
> Entries in this table remain active for `NWK_DUPLICATE_REJECTION_TTL` milliseconds and are never replaced before this timeout, so when a frame is received. [Atm14]

While the documentation does explain the option `NWK_DUPLICATE_REJECTION_TABLE_SIZE`, and refers to `NWK_DUPLICATE_REJECTION_TTL`, no hints are given on how to find a suitable value. Even the examples provided with the library do not support the developer. While the default value of the code is 10, the

examples set this value from 5 to 200. Similarly the values for `NWK_DUPLICATE_REJECTION_TTL` range from 1000 to 3000. The configuration options are set without further explanation.

The fact that the examples set different values shows that there must be good reasons to adjust these values. However, no guidance is given on how to adjust these. Such a guidance could be given by something like: "For a dense network the table size should be at least the number of nodes taking part in the network."

The average developer who wants to use the protocol without the desire to acquire a deeper understanding of the protocol itself is in a dilemma: Options are trying to understand the workings of the protocol, including the derived implications, running experiments or little of both. None of the options are very satisfying and even with a solution as presented in this work, testing all possible configurations can become tedious.

3.2.2 Changing Conditions

Even if a good configuration is found, the environment and requirements can change over time. Especially for the network layer the situation is aggravated by non-software related influences. The attributes of the network can be altered by the physical environment changing (e.g. doors being closed), interference (e.g. mobile phones, and microwave ovens), or changes in temperature. The details of these effect will be discussed in section 3.6.8, page 60.

The result of a change can improve or impair the network's properties, which in turn can improve or impair the overall QoS. Improved network properties do not automatically implicate an improved QoS and vice versa. If, for example, a connection to a certain node improves, this can increase the amount of traffic that is routed via this node, which in turn can cause an overload of that node and result in fewer packets arriving at their destination. [Wan+14]

Changing requirements normally have an external trigger and are therefore controllable. Changes in the environment, however, are difficult to foresee. Only continuous monitoring can evince these.

Independent of the source for the changing condition, the change can entail the requirement to adjust the configuration. Thus, it is often not sufficient to find one suitable configuration, but after a while it must be verified whether the configuration still performs as expected. If not, a better configuration must be found.

3.3 Optimization Strategy

To find a suitable configuration for deployment, an approach must be selected. This can be done manually, by selecting certain configurations, creating experiments and comparing their outcome. Alternatively, there are many degrees of automation that can be implemented. The first step is supporting the creation of experiments based on a parameter-set. This then allows to create a parameter space and explore it in full. Based on the results, the parameter space can be adjusted and a new exploration be started.

The process of exploring the parameter space can be optimized, for example by terminating the experiment as soon as it becomes clear that a certain requirement cannot be reached anymore. An alternative are heuristics that search more intensely in regions that yielded good results. Finally machine learning, specifically evolutionary algorithms, can provide means to automatically explore a large parameter space, finding probably not the best, yet a configuration that is superior to solutions found using other methods.

These optimization algorithms are, however, a research domain of its own. Unfortunately, most of the common algorithms are not able to cope with the requirements at hand. The fact that the input parameters are discrete, differ in range and interact, while these interactions are often not known, hinders using them out-of-the-box. Additionally the existence of local minima and the high amount of noise exclude certain algorithms. The noise can be addressed by re-running simulations multiple times and using some kind of average. This however comes at the cost of additional computation power. When running each experiment 10 times, the space that can be explored it reduced to a 10^{th}.

As the domain of automatic optimization would go beyond the scope of this work I will concentrate on tool support for manual optimization. This can then be extended by an automated process at a later point in time.

3.4 Adjusting WSN Software

When running automated tests it is desirable not to have to prepare every experiment by hand, but have tooling support to explore a parameter space. In this section I will therefore look into how WSN software and specifically Contiki can be adjusted to acquire an optimal configuration.

When testing different configurations changes can be made during two different phases of an experiment, either while building the firmware or at runtime after the node was started by sending an appropriate command to the node. Configurations made by function calls, yet having a fixed value, also fall into the category of changes that are made at build time.

When the adjustments are made at build time, this can be done at different layers. The lowest layer is making adjustments in the source code. However, most tools involved in the building process are able to handle parameters passed by the command line. This allows to make adjustments to a single build without having to revert the change afterwards.

Compile time adjustments provide little room for unexpected side effects while testing. As they are hard-coded into the firmware, they are in place once the firmware is successfully loaded onto the node.

Runtime adjustments, on the other hand, have the merit that they do not require updating the firmware, an often risky and time-consuming process outside the testbed with reliable communication using cables. To adjust the configuration at runtime, additional support on the node itself is required. The command to change the value must be detected as such and then handled by an appropriate function. Opposed to systems that provide support for high-level protocols such as TCP or User Datagram Protocol (UDP) that support to map a incoming packet to a certain application, this is not always the case in an WSN environment. When the network is closed and running only a single application, the other layers can be eliminated to the advantage of simplicity. Any incoming packet is then directly forwarded to the sole application.

Providing a general interface to adjust configurations is often not feasible. Either the value is hard-coded or subsystems might need to be re-initialized after changing the value. Thus, only such values can be updated that have previously been selected to be updatable. It must also be ensured that the configuration updates can be distributed and enforced reliably. These requirements also make it difficult to provide a generic solution to runtime updates.

Self Tuning Instead of manually setting certain configuration parameters, it is also possible to try to detect the optimal configuration automatically using an algorithm. Such elements are already part of many software components. Contiki-MAC for example tries to learn when its neighbors wake up to listen for incoming packets. pTunes [Zim+12] collects information about the whole network to then tune the network globally.

These algorithms and tools use expert knowledge about the behavior of certain components of the system to adjust the configuration. They are therefore not suitable for a generic approach. Besides, even these tools have parameters that tune them to the environment they are working in, for example how fast they adjust the system, or whether there are certain limits to the adjustments. If they are considered part of the system, even they can profit from an optimization system that is agnostic to what it is adjusting.

3.4.1 Build-Time Variability

There are multiple layers which allow adjusting WSNs during build time. Examples are Make files, header files or the source code. Some of them depend on each other, while others are independent. Each layer has its own type of variability that must be addressed separately by the test system. Most larger projects try to shift as much build time variability to the build system, allowing for a single point of adjustment, with the changes propagated to the other layers. Yet if this is not the case, or not possible, the other layers might need to be adjusted, too.

In the following I will look at the most important aspects of compile time variability and asses how they impact setting configuration options and building systems in parallel. However variability management is a research domain of its own. [Loh09; Tar13; Sin13] As many WSN systems, as well as the one I use in this work, are based on the C language and use the `make` build system I will focus on these and their common usage. Other popular solutions, most notably TinyOS [Lev+05], which has its own C-Like language and custom build system, have similar characteristics.

3.4.1.1 Build System

The build system is the central element during the creation of WSN firmware. It transfers the source code to binary code that can be executed on the micro controller. It can be very simple, only invoking the compiler, or very complex, including pre-processing and assembling the source code, before it is passed to the compiler.

There are many tools, or rather frameworks that support setting up a build system; the system itself is project-specific. It can be seen as a separate program used to generate the program to be run on the node. Due to this, it is impossible to make general statements about the build system. This implies that the availability of adjustments that can be made at the layer of the build system highly depend on its concrete implementation within the project. It also means that the build system can be extended to support additional variability.

Typical build system support passing parameters during the invocation. Based on these parameters the build system will then invoke the required tools, passing the appropriate parameters and files. For example, if the build system supports different platforms it is normally sufficient to set the target platform. The build system will then select the required compiler, the files that are needed for this specific platform, and invoke the required tools using suitable options.

Setting Configuration Options There are two main techniques to pass configuration options to the build system. One is using the command line. While this is convenient when only running a single run with this configuration, having pass all options each time is not. The alternative is to use a file. This however is less flexible, when the adjustments must be reverted. Most build systems support both methods of setting a configuration. Finally they can also be combined by passing in configuration file via the command line.

Running Experiments To find an optimal setup the experiments must be run with different configuration options. In must be ensured that each build configuration is independent of each other, and they do not influence each other. This is also important when running multiple simulation experiments, and therefore also builds, in parallel.

Instead of having one build-environment and rolling back the changes it is safer to set up a new and clean environment for each experiment. Depending on the build system this can require duplicating the environment. Although WSN systems are still pretty small compared to the available memory provided by modern computers this must not be neglected. For example a standard checkout of the Contiki-OS requires ≈25 MB. When running 4000 experiments, this already occupies 100 GB. With the files generated during the experiment it is even more.

To avoid having to duplicate all files several thousand times, the build system must support out-of-tree building. This means that all build-specific files (e.g. configuration), intermediate files and output generated by the invoked commands are saved to a separate folder, not touching the source file tree. For this to work, setting configuration options must not require adjustments to the original source tree.

Build systems that support out-of-tree builds often also allow to override files. Thus it is possible to pass a file that is used instead of the original file. The test system can then place the altered file outside the source tree without the need to alter the original file.

This however does not work well when testing different software-revisions (see section 3.4.1.5, page 48). As the build system is likely to have adjustments between two versions, it is not possible to, for example, just put the files that changed between two revisions into a separate folder. In this case each revision must be extracted to a separate build environment.

3.4.1.2 Adjusting Files

Adjusting the file is the lowest layer of adjustment. Altering files is common during the development process. Whereas adjusting source files to configure a system, is considered bad practice. It almost always has to be done manually, because automated adjustments of the code have a high risk of not doing the adjustments correctly. When altering files for configuration, it must also always be ensured that the changes are reverted before making the next change. It is therefore advisory to copy a file to another location before altering it. As already noted, this can, depending on the build system, require duplicating the build environment.

Configuration Options If files need to be adjusted, there are several options. Having only a few alternate options it is possible to create multiple versions of the file. The test system can then place the appropriate file in the build environment. The drawback of this approach is that it creates code duplications and, if for some reason the file needs generic adjustments, all versions of the file need to be adjusted.

```
#define NEIGHBORS 10
//Other code
struct t_neighbours neighbours[NEIGHBORS];
//Other code
for(int i = 0; i < NEIGHBORS; i++){ /*....*/ }
```

Listing 3.2: Replacing strings using the CPP

```
#define NEIGHBORS 10
#define DEBUG 1
//Other code
#if DEBUG == 1
printf("somevalue is %i\n", somevalue);
#endif
//Other code
#if NEIGHBORS < 8
//do not sort
#else
//sort neigbors for faster lookup
#fi
```

Listing 3.3: Enabling or disabling code using the CPP

Another method is to create a patch that describes the transition of the original file to the altered version. The test system can then apply the appropriate patch. Depending on the extent of required changes, most patch systems can tolerate alterations to the original file, while still applying the patch correctly.

Besides the classical patch systems, which support generic changes to text files, there are also more specific tools that support a more flexible approach. The most generic one is using regular expressions to replace a string possibly with one generated by the test system. Tools like `xmlstarlet` allow to alter specific file formats, in this case XML files. And finally it is always possible to write a program that does the required adjustments.

All of these solutions create files that differ between experiments and can therefore not shared between them. If files are shared between experiments it must be ensured that the original files are not touched, yet the build system uses the altered versions.

The latter can be circumvented by extending the file in a way that the build system can make the required adjustments based on its input parameters. This however requires changing the original files, yet in a manner so that they can be shared between experiments.

Making Files Configurable / The C PreProcessor (CPP) Building different variants of a software using the same code with minimal effort is a common requirement during software development. For this reason the C-Language comes with the CPP, which provides means to make string-based adjustments to the file before it is processed by the compiler.

For example the string "`#include <filename.h>`" is replaced the contents of that file. The common technique for configuration is using the `#define searchstr replacement string` which will replace all following occurrences of `searchstr` with `replacement string`. A configuration can look like shown in listing 3.2. The presented code allows to set the number of neighbors once, without the need for further adjustments.

`if`-statements can be used to enable or disable certain sections as shown in listing 3.3.

Of course the strings can also be defined in a global file that is then included by all other files using the `#include` statement. Alternatively they can be passed to the CPP during its invocation (`gcc ... -DNEIGHBORS=5`). This allows a configuration to be set manually and be passed through, derived or calculated by the build system.

3.4.1.3 Libraries

Libraries provide a set of commonly used functions. The standard C library, a set of functions that is defined in the C standard, is used by practically any project. Yet these functions can be implemented in different ways, targeting performance, memory usage or specific hardware.[1]

By default, the compiler tool chain is called with a default set of libraries. Additional libraries can be added, or the default overridden, by passing the appropriate command line parameter. Most build systems provide means to do so. Instead of exchanging a whole library it is also possible to only replace specific functions. In such a case great care must be taken that these functions have no library specific dependencies.

3.4.1.4 Tools

It is not only possible to exchange the libraries, but also the tools used during the build process, for example the compiler.[2] As the produced firmware should not differ in functional aspects, the difference between tool chains and their versions is often subtle. The quality of the generated code can however differ significantly, which particularly influences runtime and size of the code.

Under certain circumstances there can even be functional differences. For example when the exact behavior is not defined in the C-Standard[3] and the compiler implement these differently. Another example are bugs which access unassigned memory like buffer overflows. Often they have no effect, because the memory is not used, or overwritten before the next use. If another compiler tool chain produces a different memory layout, these bugs can become visible.

Most build systems allow adjusting the tool used by adjusting its configuration or by setting an environment variable.

3.4.1.5 Software Revisions

There are not only different versions of certain libraries and tools, but these also come in different revisions. The same applies not only for libraries and tools, but any part of the system. This may be the operating system, as well as the build system itself. The assumption that the newest version is the best does not always apply. For this reason the test system may want to support testing prior versions.

If there are a small number of different versions, selecting different revisions of libraries and tools can be done using the same techniques as used to select different versions. If there are many versions and these are managed using a Revision Control System (RCS) it might however be more appropriate to check out the required version as part of setting up the build environment. If the build system is part of the versioned file tree, as is the case with most WSN OS, this approach must be taken.

3.4.1.6 Tool Invocation

Not only the version and revision of a tool can influence the outcome of the final system, but also the tool-specific parameters passed to the tool. While many parameters are given by the necessary steps to build target system, there is still room for variability. Specifically optimization options passed to the compiler can have a significant impact on the resource constrained WSN nodes. In the case of optimization it is often a trade-off between size and execution speed and therefore also energy consumption. Tuning these parameters can therefore cause the final binary to not fit on the node anymore or to fail to meet timing or energy constraints.

3.4.2 Run-Time Variability

Opposed to compile time parameters, run-time parameters can be adjusted without the need of loading a new firmware on to the node. The drawback is that one must ensure that the node is configured to the value expected. This can be done by adjusting the compile time default value, losing the advantages of run time adjustments. Otherwise it must be done by setting the value after each reboot, either using the network or by saving it in a non-volatile configuration space.

[1] Examples for the standard C library are GNU C Library (glibc) (`https://www.gnu.org/software/libc/`), µClibc (`http://www.uclibc.org/`), musl libc (`http://www.musl-libc.org/`) and diet libc (`http://www.fefe.de/dietlibc/`).

[2] Common compilers for the C-Language used in the WSN context are GCC (`https://gcc.gnu.org/`), SDCC (`http://sdcc.sourceforge.net/`) and IAR Embedded Workbench (`https://www.iar.com/`).

[3] A typical example is a right bit-shift of signed values. The C-Standard defines it as implementation specific whether to shift 0 or the most significant bit.

Changing a parameter can impact values that enable or disable paths of code. Values can normally be migrated to run time parameters, by adding a variable that can be adjusted at runtime. For alternative implementations of functions this is not always possible. At the cost of a variable and some additional program code, small features often can trivially be placed in an if-clause (listing 3.4).

if (para == 1) {/*feat1*/} **else** {/*feat2*/}

Listing 3.4: Changing a configuration using if-else

Large changes, for example the replacement of the Medium Access Control (MAC) protocol are often not as trivial. Although the mac layer is supposed to be somewhat transparent to the higher layers, this is not always the case. Often higher layers adjust themselves to the lower ones and can therefore not cope with a change at run time.

Another problem arises when using alternate implementations. These often have the same function names and therefore cannot be distinguished from the calling code. In such a case including both versions in the same binary is not intended by design.

All in all, adding two alternative implementations always requires additional resources. The impact depends on how much code is added to also provide the alternative code path. Whether it is worth migrating a compile time option to a run time option is therefore highly situation dependent.

3.5 Executing Experiments

Ideally the experiments are run using the target environment, as this provides the most accurate results. Yet this is often tedious at best. One must not only acquire and collect data, but also distribute different configurations. If two configurations are incompatible one must not only ensure that the configuration has reached every node, but the switchover must happen somewhat synchronously. Sometimes this can be done by adjusting a variable, but often it requires updating the firmware, as the configuration is hard-coded into the code.

Just running an experiment is not sufficient. Data must be raised, collected and analyzed. Ideally this data also helps comparing solutions with regard of the optimization goal. Otherwise it is worthless. Due to the wireless nature of WSNs, they also have a strong component of randomness. Experiments must therefore be executed multiple times and analyzed statistically.

In the following I will look into the aspects of running experiments in general, on real hardware and using simulation.

3.5.1 Evaluating WSNs Experiments and Its Limitations

Being able to change a configuration is only half of the story. There must also be means to measure whether the changed configuration has the desired effect or not. Also WSNs are prone to random effects, specifically packet loss and changes in the link quality and therefore also topology. Thus it must also be ensured that the observed effects are actually caused by the change, rather than a random effect.

Secondly certain data can and must be acquired indirectly. Waiting until a node fails to see whether the energy saving change has an effect, is normally not an option. Also adding additional hardware is not suitable either. Instead metrics, for example the time micro controller and radio spend in certain states can be used to estimate the energy consumption.

As the actual metrics required to evaluate a configuration are application specific, I will look into the general aspects and information that can be provided by any system.

3.5.1.1 Acquiring Data

The first step to evaluate a WSN is to acquire data to understand what is going on inside the node. There are different options to do so. They differ in whether it is possible to access the information, the effort to collect the data and the impact on the system, the so called probe effect.

The first requirement is of course that the information is accessible. Ideally this is some variable that is accessible anyway. It can also be of interest how often some event happens. Such events typically execute certain code, which can be augmented with a counter. For example, by increasing a counter each time the `send` or `receive` function is called, the number of packets that are sent and received can be monitored.

The number of packets dropped can be acquired using the same technique. Of course also values that need some logic like the maximum level of a buffer can be saved in a variable for later retrieval.

Many micro controllers also provide information that can be interesting to evaluate an experiment, for example the cause of the last reboot. Typical causes can be manual reset, watchdog or a power outage. Radio chips also provide additional information, which can be queried. A common value of interest it the Receive Signal Strength Indicator (RSSI), reflecting the energy received by the antenna (see section 2.6.5.2, page 30).

Sometimes indirect methods are required, for example the runtime of a function or the time spent in sleep mode. These can be acquired by reading a timer register when entering or leaving a certain state. By aggregating these values it is possible to get a pretty accurate estimation of the actual power usage [Hur+11].

Finally, there is information that requires additional hardware. A voltage divider to measure the battery voltage using an Analogue Digital Converter (ADC) is a common example. Accurately measuring the energy usage on the other hand is more complicated and requires external hardware [HWS12; Hön+14]. Similarly if accurate timing of Input / Output (I/O) lines is required this can only be done by using, for example, an oscilloscope.

Interfacing Data Acquiring the data is only the first step. It must also accessed to be evaluated afterwards. This can be done at the end of an experiment, or continuously, creating a time series, to allow for a more accurate data or when monitoring long running system.

The simplest method of retrieving data is also known as printf debugging. Each time an event happens or a value is read it is printed to the serial. Outside a testbed there normally is no serial interface and the information must be sent wirelessly. If no serial interface is available and wireless communications should not be interfered, the information can also be saved to non-volatile memory. Common are external flash memory and SD-cards as these provide a simple interface that can be used by micro controllers without too much overhead.

Sometimes it is also reasonable to redirect information. For example by toggling an I/O pin each time a function is entered. An oscilloscope can then be used to analyze function calls. System states are also often visualized using LEDs (Light Emitting Diodes), and can therefore be interfaced with the eye.

Finally, there are special interfaces designed for debugging, such as a JTAG (Joint Test Action Group) interface. These can also be used to extract information, for example by monitoring a variable or counting how often a certain command is executed. The amount of features provided by such interfaces depends on the capabilities of the controller as well as the tool. All of these tools allow halting the controller and reading memory and registers. Being able to interrupt the execution when passing a certain program address (breakpoint) is also supported by most debuggers. More powerful debuggers provide more breakpoints as well as stopping on memory writes and reads. Some debug interfaces even provide the ability to trace the program execution without influencing the execution itself. This feature however requires quite a few I/O pins, and can therefore only be found on micro controllers that are more powerful than those used on typical WSN nodes.

Aggregating Data The raw data can quickly exceed the resources available and must be aggregated. For example printing to the serial for each packet sent can cause a significant amount of data. And sending a packet to the sink for each packet sent is obviously not reasonable. For simple events like sending and receiving packets it is often sufficient to increase a counter. For information like the RSSI, things are a little more complicated. Values like the minimum, maximum and average can be calculated continuously quite easily without the need to save the actually sampled data, which is quite important due to the scarce resources. This is also possible for the standard deviation or rather the variance [Wik15a], not however for the median [Cha10].

When collecting data, care must be taken that the data is statistically valid. For example, due to the non-Gaussian distribution of RSSI values the informative value of average and variance are limited. The appropriate method of aggregating data and its explanatory power is however a statistical, rather than a technical issue, and will not be further discussed here.

Probe Effect Most methods of monitoring a system influence the system in the one way or the other. Even the minimal delay required to increment a counter, will let the systems with and without counter behave slightly different. Most likely the effect will not be visible in practice, though, at least when the counter is incremented seldom enough.

The effect that is caused by monitoring a system in a way that will not be done under normal operation is called probe effect. The impact of the probe effect depends on the required resources. Additional memory

requirements can cause other functions to fail, because they are unable to allocate memory or even cause stack overflows.

In most situations however the effects of the additional execution time are more visible. Especially the network imposes real-time constraints on the system. If the buffer is not read in time it will be overwritten by the next packet – or the next packet is dropped, depending on the implemented logic. Thus, if the collection of data costs too much time, this can cause a severe degradation of the system. On the other hand, slight delays can shadow concurrency issues so that these are only visible without the monitoring in place.

While the impact of incrementing a counter is marginal and can normally be neglected, calling a function like `printf()` can have a significant impact. The formatting function itself requires 2079 B of program memory[4]. It can also have a significant influence on the execution timing. One reason is that `printf()` needs to parse the format string and format the passed parameters accordingly. Especially when formatting decimal numbers performance degrades because most micro controller used in WSN do not have a hardware division unit. The division must therefore be done in software, which also varies in execution timing depending on the input data.

The second reason is more generic as it does not only apply to `printf`, but to any function writing to the serial. Often the hardware provides a fifo-buffer of a few bytes (typically 4 or 8) and it is also possible to have a larger second-level buffer in software. If however the buffer is full, either because the string is too long, or the rate at which the data is written is too high, the printing function must block until the buffer has free space again. The alternative is to drop data, which often is not much better.

Obviously the serial interface is not the only hardware component that can slow down a system. Examples are internal or external flash memory or SD-cards. The speed of accessing external memory highly depends on the bus used to interface it. Although many micro controller support attaching external memory using a parallel bus[5], external flash is typically interfaced using a serial interface, to save the pins and simplify the layout of the circuit board. The communication is slowed down by the serialisation of the data and the overhead of the communication protocol, which requires additional functions in software and increases the amount of data to be transmitted.

Even special interfaces like JTAG that are non-intrusive to the code and memory, do influence execution timing. The principle is the same as for the serial bus. As the number of pins for the debug interface are limited, the data must be serialized. Although the micro controllers contain a built-in debug unit that can monitor certain actions, the controller must be stopped to transmit the data to the external hardware. Only more complex debug interfaces with many pins do allow real-time monitoring of the execution without influencing the execution. The external hardware will however influence the characteristics of the antenna and have at least a slight impact on the network.

3.5.1.2 Interpreting the Data

Data must not only be collected from the nodes, but also interpreted. As noted before, the statistical analysis of the data is beyond the scope of this work. There are however two aspects worth noting: Correlating data within the time space and the layer at which the data is interpreted.

To determine how long it takes for a packet to reach the sink or which node has sent its packet first, the points in time when these events happen must be known. To get these points in time, it is first necessary to correlate the data within the time-space.

When using simulation this is rather easy, because most such events can be logged by the simulator. When such events are logged to the serial, multiple effects can distort the results. First of all, the time it takes for a message to be printed to the serial depends on the filling level of the buffer. If it is empty the message is printed right away, if it is full it may take some time. Secondly additional jitter is added on the receiving side: The serial has multiple buffers in hardware and the OS, and the receiving program must be scheduled before it can process the date. If the data is collected by multiple computers intermediate processing and the network can also add a bit of delay and jitter.

If the nodes have synchronized clocks, this can be addressed by adding timestamps to the events. However, synchronizing the clocks requires additional software infrastructure and network overhead. Additionally, the time stamps increase the amount of data that has to be printed.

The second issue, the layer at which the data is interpreted, is for example of interest when the routing is not predicable. This is most often the case with most routing algorithms that support discovering the route. The route then depends on which node reacts faster or on the signal strength of the answering

[4] Based on the Debian packaged msp430-libc version `20120224-1`

[5] An example it the BTnode [BKR03], which has 1 MiB of external SRAM.

packet. Consequently, a node that has to forward a lot of packets in one experiment, might have to forward none in the next, although the environment did not change.

Data like packet loss should therefore not be interpreted at a node level, but at a network level. Even if the routing algorithm is designed to balance the load of the nodes, the data must always be interpreted in a global scope. Of course there are always exceptions, for example when nodes play a special role.

3.5.2 Testing Configurations

To test configurations one has several options. While results from the actual deployment, opposed to the testbed, obviously have the highest accuracy, acquiring them also requires the highest effort. There are multiple reasons why a deployment is unsuitable for testing. First of all, one normally wants to use the deployment productively and not for testing. Secondly, when testing, the nodes must be updated using Over The Air (OTA) programming or gaining physical access. The first presents a high risk of bricking the node, while the latter which can be quite time consuming, depending on where the node is located. Even though, to get the maximum performance, some adjustments are normally required after the deployment.

The second option is a testbed. It normally does not resemble the deployment in terms of topology and radio characteristics, but provides reliable means of programming and interfacing the nodes. Reliable communication is extremely important when running automated tests. Having a node running the wrong firmware can render the results of the experiment worthless. Also collecting results using a poorly working network configuration can become infeasible, and switching to a well known working configuration after the experiment adds a significant overhead.

While a testbed is suitable to thoroughly test the hard- and software, it does not resemble the topology (i.e. different number of nodes, connection quality, number of connections, etc.). Great care must therefore be taken when making network-related assumptions. For example, the connectivity between nodes may be better or worse in the final deployment and also the typical number of packets that must be forwarded in a multihop network, and therefore the required bandwidth is very likely to differ. A system that runs perfectly in a testbed can therefore horribly fail running as deployment.

Finally there are simulations. They were already presented in section 2.4, page 20. For testing different configurations they have multiple advantages. Most importantly, simulations can be run in parallel. The lower bounds for the average time to test a configuration is therefore only limited by the available processing power. For this work I was able to simulate several thousand 20 min experiments within a few hours – 1000 20 min experiments in a real testbed take more than 2 weeks. Reprogramming the nodes, especially when using OTA programming in a deployment, can easily take several minutes and increase the duration even more.

The second advantage of simulation is that it is possible to test multiple scenarios. One is therefore not limited to one or two testbeds, but can “create” new environments to see how a system behaves, for example with many nodes or high congestion. The high control over the simulation also allows to reproducing experiments, which is more or less impossible with real nodes, as there are too many random effects.

Of course simulation also has a disadvantage. Although some parts of the WSN, specifically the micro controller, can be emulated (see section 2.4.2, page 21), especially the networking part can only simulated accurately to a certain degree. The quality of the model depends on many factors, like the complexity of the model, but also the quality of its configuration. In the next section I will therefore look into the realism of simulating WSNs.

3.6 Realism in Simulation

When analyzing the realism of a model used by a simulator, the core question is how much realism is needed to draw the necessary conclusions. For example, to make a general assessment of a network protocol, it is unreasonable to take the burden of accurately simulating the whole node. If this protocol is rather complex, but is supposed to run on WSN nodes, it is necessary to verify it using an instruction level simulator. Otherwise, it might well be that the node is not powerful enough to run the code or the radio does not meet the requirements. In this case using a simple network model is justifiable, if the objective is to make a statement whether that node is capable of running that network algorithm.

To select a suitable model and configure it correctly, detailed knowledge of the target environment is required. In the following exemplary characteristics found in literature are presented. The effects will be discussed in detail afterwards.

Woods / Jungle [Cer+10] The wilderness is an ideal environment for WSNs. There is practically no interference and little changes in the environment, besides temperature and humidity. The range mainly depends on the vegetation and the undergrowth as these influence signal dampening and reflection.

Permafrost [HIT08] In a permafrost environment there are likely to be significant differences in temperature depending on whether the node is exposed to the sun or not. As these can be 10s of °K the clocks of the nodes run at a different pace (see section 3.6.5, page 58). This is especially problematic when nodes sleep for a longer duration and do not wake up at the same time to exchange radio messages.

Industry / Greenhouse [Odo+13; Sur+13; Yoo+07] The industrial environment provides a relatively controlled environment with few changes. At least as long as there is little interaction with people that introduce additional communication devices or move around a lot. Radio interference is mainly caused by other wireless networks or machines. As the other radio networks are under ones control, these can be configured to minimize interference. Also there usage scheme is known, and the WSN can be adopted to it. The same applies for the machines. Their radiation pattern is known, and changes little over time. The connection ranges mainly depend on the surrounding infrastructure.

Office [Ami+13; Str+14] An office environment is very challenging. There are many changes in connectivity caused by people walking around, using radio devices, opening or shutting doors and windows. There is also a high amount of interference, as mobile devices communicating wirelessly are ubiquitous. Opposed to the industrial environment, there is hardly any control over them.

These examples show that the requirements in regard to the simulation model differ significantly based on environment of the deployment. The exact placement, but even the orientation, can have a significant influence on the connectivity. When running an experiment it must be clear that the results can only apply to the setup used. To make a generalized statement it is not sufficient to run an experiment with a model that reflects, for example, a specific office environment, but it must be configured to different office environments.

A simple, but well configured model, can yield better results than a sophisticated, but badly configured model. Yet even with the best simulation model that is correctly configured, it is dangerous to make assumptions about the realism of simulation without verifying that the assumptions made are correct. For example, the developers of the TOSSIM WSN simulator did not implement write protection in their flash memory model. Probably they assumed that, as the flash memory cannot be written directly, nobody would do so. The developers of a software update system tried to overcome memory limitations by also using the flash memory to save additional data. [PB09] As they used the simulator for their development, this did work. They successfully published their results at the INFOCOM conference 2009, which had an acceptance rate of ≈20 %.

In section 2.4 (Simulating WSNs, page 20) I already presented the different types of WSN simulators. The rest of this chapter will analyze the different components of WSN simulators and how they influence their realism.

3.6.1 Monte Carlo Simulation and the Pseudo Random Number Generator (PRNG)

Simulators are, opposed to testbeds, able to reproduce results, as they are able to control every aspect of the modeled environment. Random effects like packet loss are reflected by using as PRNG. By initializing the PRNG with the same seed, the PRNG outputs the same stream of seemingly random numbers and the simulator will reproduce the same simulation run. This kind of simulation, which is based on random decisions, is called Monte Carlo Simulation, referencing the casino in Monte Carlo, Monaco.

Impact When choosing the wrong seed, there is a very low, but existing chance that for a connection with a PRR of 90 % the first 100 packets get dropped – or none.[6] Any of those two cases can lead to

[6] Theoretically the chance is 10^{-100} for real randomness, but PRNG have a limited number of seeds they derive their values from (typically 2^{32}, 2^{48} or 2^{64}). It is therefore possible that this scenario cannot be created at all – or is more likely to happen than in the real world. [PJL02]

wrong conclusions about the general behavior of a certain software version. The results of Monte Carlo Simulations will spread and simulations must be repeated multiple times using different random seeds to get credible results. Due to its stochastic nature the results of Monte Carlo Simulations must be, like real experiments, evaluated using statistics.

Discussion In [PJL02] Pawlowski et al. survey over 2000 networking publications published in journals and on conferences between 1992 and 1998. They find that the majority of publications using simulated randomness neither discuss their PRNG nor statistically analyze their results. They therefore announce a *Credibility Crises* for Simulation Studies of Telecommunication Networks. Kurkowski et al. come to similar results for the Mobile Ad hoc NETwork (MANET) domain, analyzing publications at the MobiHoc conference from 2000 to 2005. To my experience the criticism of missing statistical analysis by Pawlowski et al. also applies to the WSN domain as of today.

The other major criticism voiced by Pawlowski et al. is that not enough care is taken when selecting a PRNG. Kurkowski et al. however do not think this is a real problem. Both state that the model using a PRNG must be validated and verified. In my view, the issue of validating models is currently more pressing for WSN domain than the quality of the PRNG. This, however, does not weaken the point that the stochastic properties of Monte Carlo Simulations must not be neglected.

3.6.2 Node Start-Up Order and Inter-Node Offsets

Using a simulator, it is possible to start nodes at fixed offsets, yielding repeatable results. With real nodes this is not possible, at least not without a marginal jitter that can make the difference between successful transmission and collision. In the typical testbed setup, where the nodes are connected via USB-to-Serial adaptor, reducing the accuracy to a few 100 µs can already be challenging. Doing so wirelessly is possible, but requires clock synchronization, which can become tedious. [YK14] Simulation therefore has an advantage over real experiments, yet the start-up offsets must be derived from a random source and spread over a reasonable period to allow a realistic Monte-Carlo-Simulation.

The effects of inter-node offsets of periodic events are not only hard to reproduce, but also hard to detect. A simulator can visualize the state of the network and help to detect these issues. Figure 3.2 shows the Cooja Timeline [ÖED10] for the Cooja Simulator. In fig. 3.2a, which depicts the situation when all nodes start at the same time, one can see that the RDC protocol wakes all nodes at the same time to probe the channel (gray). Whereas in fig. 3.2b, where the nodes were booted over a period of 1 s, the points in time at which the radios probe are more distributed. The regularity of the wake-up pattern is still visible.

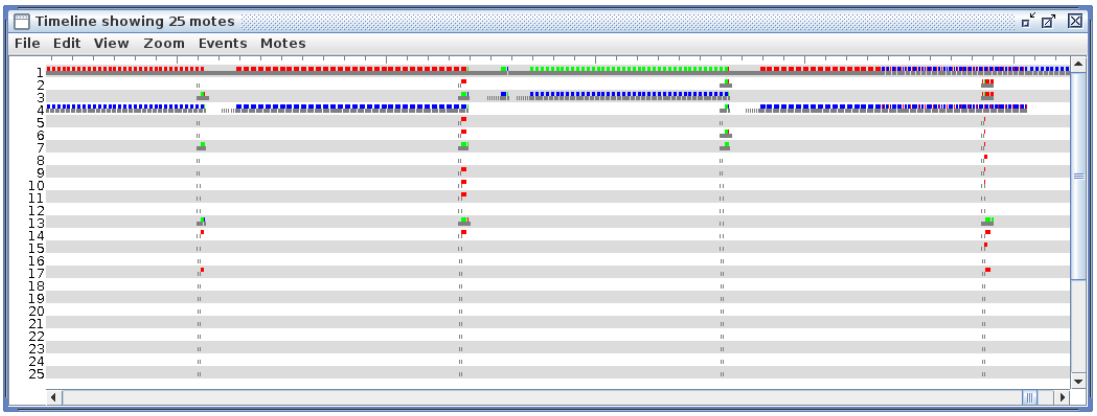
When using a real network, getting access to this kind of information is not easy. One option is to use a sniffer as presented in [ZHS12]. Yet such a sniffer can only see the state of the network at the point it is placed. Alternatively the state of the radio can be either dumped via a side-channel or saved for later analysis. This however introduces new problems, such as probe effects or hidden delays due to buffers (see section 3.5.1.1, page 49).

Impact For WSNs the start-up order is normally of minor importance, because this is expected and software is designed to cope with this. The offset between periodic events, however, can make a big difference. A typical example for this are nodes sampling and sending sensor data at a regular interval of, for example, 60 s. If the sending of the packets is evenly distributed over the minute, the network load will stay low, whereas if the offset between sending packets is low, all nodes will send their data at more or less the same time. This will cause a high load, which can result in high packet loss. Without some kind of synchronization between the nodes, they will keep to their schedule and are therefore unable to recover by themselves. When simulating nodes this issue is aggravated, especially without clock drift that will eventually change the offset between periods of different nodes (see section 3.6.5, page 58).

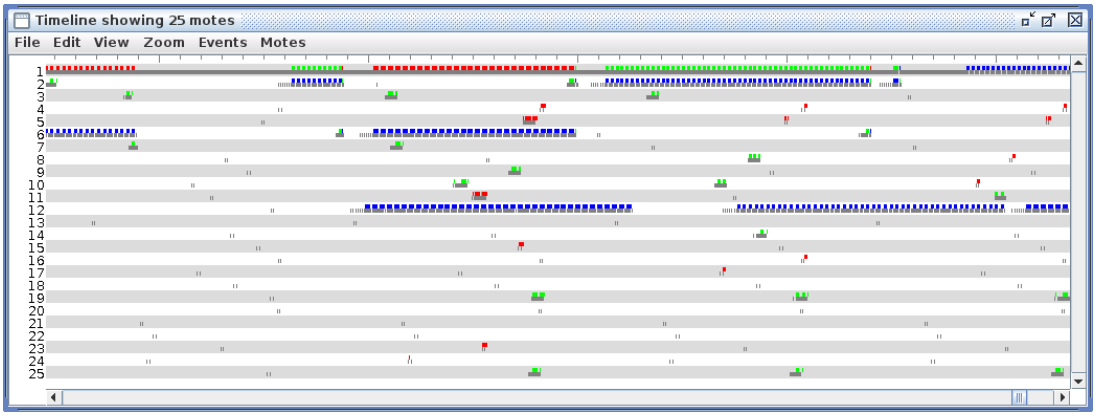
Most of the effects caused by inter-node offsets become visible at the network layer, as this is where the nodes interact. They are not necessarily triggered at this layer, as the example just given shows.

To show the severity of the issue, especially when trying to reproduce results using real nodes, I will discuss the effect of inter-node offsets on CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) mechanisms.

Assume the two nodes 1 and 2 in fig. 3.3 send data to the sink S with the same frequency. In fig. 3.3a the two clients can only communicate with the sink, but not with each other. If these nodes have an inter-period that causes their packets to interfere they have no way of recovering. They cannot detect the collision as they cannot send and receive at the same time. The only method of noticing the collision is the sink not sending an acknowledgment, which can also be caused by the sink simply being out of reach.



a) Node start-up at the same time



b) Node start-up randomly distributed over 1 s

Figure 3.2: Cooja's Timline showing the network state of 25 nodes. When a radio is listening, this is shown in gray, sending in blue, receiving in green and collisions are red. Due to the radio model used in this example, connections considered too far away for successful reception, but close enough for interference are also marked as interfered.

The clocks of real nodes always drift a bit and they can therefore recover eventually (see section 3.6.5, page 58)

Figure 3.3b shows a scenario where the nodes can avoid collisions using CSMA-CA. Yet even here a minimum offset is required to mitigate the problem described and CSMA-CD (CSMA with Collision Detection) is not possible using the simple low energy radio chips.

Figure 3.4 illustrates a typical timing for such a situation. Two nodes 1 and 2 start sending their data at an offset T_{off}. The radios first check whether the channel is clear by turning on the receiver. If the radio detects a signal the transmission is canceled. As the channel is clear, node 1 starts sending. Switching from receiving to sending takes a certain while (T_{sw}).

Node 2 also checks whether the channel is clear. Although node 1 is already sending the overlap ($T_{overlap}$) is not long enough to detect the signal. It therefore also starts sending. If the signal at the receiver is similar in strength, neither can receive data.

a) Clients can only hear the sink **b)** All nodes can hear each other

Figure 3.3: Very basic networks to explain issues with collisions.

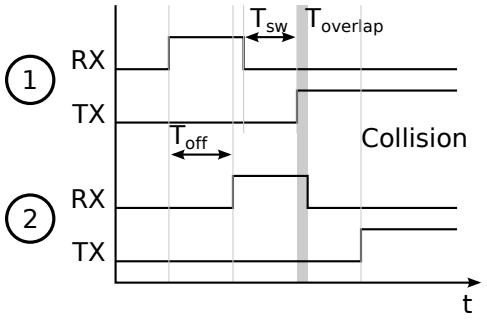


Figure 3.4: Although nodes start sending at an offset, the overlap between sending and sensing is not sufficient to avoid a collision. The high level represents RX or TX being active. T_{sw} is the time the radio takes to switch from receiving to sending. $T_{overlap}$ highlights the time when node ① is already sending while node ② is still listening for potential traffic. It is marked in gray. Because $T_{overlap}$ is too short for node ② to detect node ① sending, it starts sending, too. With both nodes sending at the same time, they interfere each other and neither can be received by other nodes.

The minimum offset T_{off} required to avoid a collision depends on the time needed to switch from receiving to sending (T_{sw}), as well as the minimum time required to detect the signal. The time it takes to switch from RX to TX depends on the radio chip. For the CC2420 this is at least 128 µs. [Tex04] The minimum duration it takes to detect a signal ($T_{overlap}$) depends on the radio chip, signal strength, signal quality and other factors like background noise (see also section 2.6.7.3, page 35). Kiryushin et al. have examined this experimentally [KSM08].

RDC mechanisms are often vulnerable to similar effects and offsets between the wake-up schedule of nodes can have a huge impact on the behavior of the network. Carrano et al. discuss the domain in detail. [Car+14] Other protocols that are TDMA (Time Division Multiple Access) based need to synchronize their schedules actively. The impact of the boot-up offset is therefore limited to the synchronization phase.

It should be noted that the presented effects are strongly influenced by the radio chip. Whether a collision is avoided in simulation therefore not only depends on the offset, but also on the radio model and on the correct implementation of the radio chip.

Discussion The severity of the issue is often neglected and not taken into account during system design. Neither at the network level – none of the network protocols provided by Contiki explicitly take unsuitable offsets into account – nor at the application level. I see two reasons for the missing apprehension: Firstly, if the testbed misbehaves, the experiment is repeated and suddenly everything works perfect. Instead of being a conceptual flaw, the misbehavior is considered a random effect (e.g. interference by other devices) that cannot be traced back due to missing data. Secondly, clock drift, an often unwanted effect, can fix the issues described, as it changes the offset over time. Sometimes it is even considered as solution to the problem. [Sze+04] Yet clock drift only helps with long running experiments.

3.6.3 Simulator Abstraction Level

The different simulator abstraction levels were presented in section 2.4, page 20. Choosing the abstraction level of the simulator is often a trade-off between run-time and accuracy. Instruction Level Simulators (ILSs) support accurate simulation by interpreting the binary code that is programmed to the target, but are currently limited to simulating up to about 100 nodes in real time.[7] Network Level Simulators (NLSs) require the code to programmed against their API (Application Programming Interface), but support efficiently simulating large networks over long periods as the code is compiled for the native platform.

Finally System Level Simulators (SLSs) are somewhere in between those two. They provide an interface that is similar to a typical Hardware Abstraction Layer (HAL). This allows adjusting an OS interface to the SLS instead of the real hardware. While the code is still compiled for the native platform, it uses the API of the OS running on the target platform. The application code can therefore be compiled for the target platform without the need for adjustments.

Impact When using NLSs, almost all limitations imposed by the target platform are lost. If at all, NLS are able to reflect the behavior of the radio chip. While this allows rapid prototyping of new protocol ideas, NLS can make no statement whether it is possible to use this protocol on real sensor nodes.

[7] The actual number depends on the type of node and the code running on the node. If the nodes spends a lot of time sleeping less processing power is required, speeding up the simulation.

Using a SLS it is at least possible to evaluate a concrete implementation of a protocol and the application using it, as the implementation uses the API provided by the OS. It can still not make a statement whether the resources are sufficient. Especially for network protocols this does not only apply to computational power and memory, but also to the communication channels and the model of the radio chip.

Even if it is possible to run the same code on real hardware, the inaccurate timing can have a significant impact on the outcome. [GCM11] At least in terms of runtime, this can be addressed by annotating the code with timings required to execute it on the target platform (see section 2.4.3, page 21, [Mös+13]). The same approach can be used to estimate the memory usage.

Only ILS can provide accurate emulation. They do have the major drawback that the accuracy comes at a high cost: The resources required by the simulator are much higher.

The problem domain of concurrency issues can only be reflected by ILSs. Concurrency issues especially arise when the normal program flow is interrupted by an Interrupt Service Routine (ISR). One of the properties of interrupts is that it is almost always impossible to predict where the code is interrupted. As long as the ISR does not interact with data structures that are currently processed by the interrupted code, the only effect is a delay in the execution of the main code. If both, the normal program flow and the ISR, access the same data, unpredictable effects can be occur.

A typical example for the WSN domain is an ISR copying data from the radio chip to an internal buffer, and a normal task processing the data. If the interrupt is triggered again before the task has finished processing the data, it will overwrite the data. As the task has no simple way of detecting that the data was altered it will first process the old and then the new data, which is likely to result in a corrupted result.

While the example with the buffer is still relatively easy to detect, things become less obvious using simple variables. The same effect can be triggered when accessing a variable that is larger the bit-width of the used architecture (e.g 16 bit variable on a 8 bit micro controller or a 32 bit variable on a 16 bit micro controller). (Example[8]). As most native systems have a 32 bit or even 64 bit architecture, it is more or less impossible to trigger this effect using a SLS or NLS.

Another aspect that can only be reflected by ILSs are platform specific attributes. Many platforms that support more than 8 bit, only support aligned reads and writes. For example 16 bit words can only be read at even memory addresses. The result of reading or writing an odd address is not defined, yet most of these platforms do not provide means to detect such issues.

Discussion While NLSs provide an easy way of prototyping new network protocols, they are not suitable for anything that goes beyond a conceptual evaluation. Using a SLS, especially one that takes the runtime of the code into account, does improve the result, as at least the concrete implementation can be tested. Yet to get a sound estimation on whether the software can be run on the target node, an ILS must be used.[SK13]

Using an ILS does not relieve one from testing the code on real hardware. Minimal inaccuracies, for example in timing, can make a big difference when interrupts are involved. Also, as discussed, ILSs are seldom feature complete or have inconspicuous bugs that only show under certain circumstances.

3.6.4 Digital Hardware

As already hinted at in section 2.4.2, page 21, the rather simple digital hardware components used in WSNs can normally be emulated or at least be simulated very accurately by ILS. This does not apply to more powerful hardware with features like caches and instruction pipelining, which, as of now, is rarely used in WSNs and not in the scope of this work. In the future this may however change. Also, all other digital parts of a typical WSN node are deterministic and can be emulated without the need of a PRNG. Therefore, when only considering these components, a Monte Carlo Simulation is not necessary.

Impact Issues with digital hardware root in unimplemented functionality or bugs. This is an implantation specific, but not a general issue.

Discussion Opposed to NLS, where it is clear that the simulation does not perfectly reflect reality, the realism of the simulation/emulation holds the danger of trusting it too much. As discussed in the introduction of this section, bugs in the implementation can lead to false conclusions about the real hardware. Emulation therefore does not free the developer form verifying the system on real hardware.

[8] A 16 bit variable on an 8 bit micro controller has the value 0x00FF. The memory can only be access byte-wise. The main program reads the lower Byte (0xFF). An interrupt increases the value by one (0x0100). The main program reads the second byte (0x01) and assumes the variable has a value of 0x01FF, which is not near 0x00FF nor 0x0100.

Using simulation to emulate the hardware also has a big advantage. It is possible to highlight problems that not visible on real hardware. For example when doing unaligned reads and writes the behavior on real hardware is often undefined. However, no interrupt is triggered and the execution continuous. Typically the impact of such bugs is seen much later. A simulator is able to detect such issue and inform the user about them. Other examples are divisions by 0, which always returns 0, or stack overflows. There are means to detect both of them, but are not used due to the additional overhead.

3.6.5 Clock Skew and Drift

WSN nodes need some source to drive their clock. There are two common techniques. The first is using a crystal or ceramic oscillator, as used in digital watches. The second is a RC circuit. It works by charging and discharging a capacitor (C) using a resistor (R).

The RC circuits are very cheap and have a fast startup, thus take little time to generate a stable signal after being turned on. They do however lack accuracy. The skew (offset between expected and actual frequency) is highly influenced by the production process, temperature, humidity and voltage. [max] They must therefore always be calibrated if they are supposed to yield accurate timing.

Crystal or ceramic oscillators which are more expensive and have a longer start-up-time, but are much more accurate and are only slightly influenced by the environment. A typical clock skew of a crystal oscillator used in WSN is ± 20 to ± 50 ppm[9] at 20 °C. Also changes in temperature only have an influence of about ± 10ppm over a range of about -45 to 100 °C. [Vig00] In comparison, the internal RC-circut of the micro controller used on the Sky node has a typical temperature drift of about $-0.38\,\%/°C$. [msp] Yet random effects can still cause short term changes to the frequency that go beyond the specified accuracy of the quartz [Vig00] Therefore, unless synchronized by an external source, clocks of the nodes always drift apart.

The energy consumption of both sources is more or less proportional to the frequency. [max] Most micro controllers therefore support multiple clock sources. A high frequency source, which is used when active, and a low frequency source, which is used to drive timers and components that need to be kept on even in sleep mode. A common solution is therefore to use a low frequency crystal oscillator to have an accurate long-term timing and a RC-circuit to drive the micro controller in active mode. The RC-circuit can be calibrated using the crystal oscillator.

As long as there is only one clock in the system and no real time requirements this can safely be ignored. If timing is critical, for example when controlling a cyber physical systems, clock drift must not be neglected. The same applies to the clock skew cause be two clocks (on different nodes) not running at exactly the same pace.

Yet even instruction level simulators with WSN support like AVRora, MSPSim and Cooja[10] lack support. Most other simulators only support constant drift, if at all, although this has been found to be an issue with simulators by multiple researchers. [YNB11; PPB07] Very specialized simulators like Castalia do have more complex models that support non-constant drift. [FMT12]

Radio chips are normally self-calibrating and do not directly use a crystal oscillator as clock source. They can therefore compensate changes in temperature. Besides, the effects caused by frequency drift cannot be distinguished from those caused by noise without an effort. When simulating the radio network, ignoring clock drift is normally a valid option because it is hidden by other effects.

Impact Any system that derives events from a clock is influenced by clock drift. The impact depends on how much the system relies on the accuracy of the clock and whether it interacts with the outside world. If the node solely writes sensor data to internal memory, clock drift can often be ignored. This is not necessarily the case for metering tasks, as the drift may result in incorrect measurements. Especially when the data is integrating over time. The effects can usually be seen at the network layer and are caused by inter-node offsets of periodic events. These have been discussed in more detail in section 3.6.2, page 54.

How much a network is influenced by clock drift changing inter-node offsets and how much this influences simulation results depends on the protocols used. TDMA protocols, for example, must be able to cope with clock drift and regularly adjust their inter-node offsets. They are therefore rather robust against clock drift. Thus the lack of simulated clock drift reduces the amount of synchronization needed, but the impact on the results should be tolerable. This assumes that the synchronization is working, and clock drift is considered within a certain tolerance. Of course no conclusions about the synchronization mechanism can be derived from the simulation without simulating clock drift.

[9] ppm = parts-per-million = 10^{-6}

[10] As of 2015-05-13 Contiki has support for constant drift (`9261ff5d` [Con]). This has however not been evaluated.

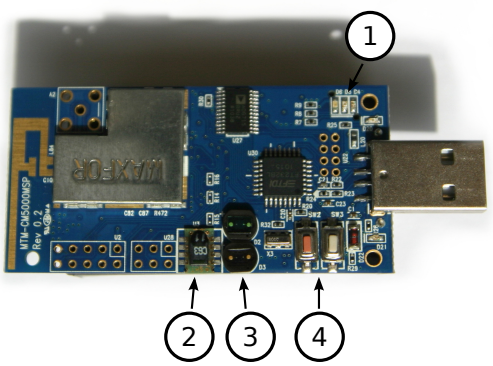


Figure 3.5: Sensors and actors of the Sky mote: ①: The colored LEDs to show display information. ②: A temperature and humidity sensor. ③: Two light sensors. One with an infra red filter. ④: Two buttons. One to reset the node and one to allow user inputs.

For protocols that do not have explicit synchronization, clock drift can have advantages as well as disadvantages. In [Sze+04] nodes sent out data at regular intervals. If they did so at the same time this can result in a collision despite CSMA (see section 3.6.2, page 54). Due to clock drift the nodes eventually drifted apart far enough to avoid the collision. Of course this will, sooner or later, also introduce new collisions. The absence of clock drift in simulation can therefore lead to flawed conclusions. If the initial inter-node offsets are unsuitable, the network will never recover, while if they are suitable, it will never degrade. A more comprehensive discussion of the issue can be found in [Gan+09; Kel+11].

Discussion Not being able to simulate clock drift must be considered a flaw of many WSN simulators. It is not only not possible to detect problems with synchronization mechanisms, but also neglects the great advantage of simulators being able to simulate any clock drift. In harsh environments with large differences in temperature like glaciers, clock drift cannot only become relatively strong, but also change within a short time when the node is exposed to the sun [HIT08]. It requires some effort to reproduce such an environment in a testbed.

Which clock-drift model to use once again depends on the target deployment. As shown by Szewczyk, Uddin et al. [Sze+04; UC10] clock drift mainly consists of a constant value plus random noise. By providing means for constant drift or also adding some randomness as suggested by Ferrari et al. [FMT12] could significantly improve the realism of simulations for most environments. Yet even changes in temperature can be modeled with reasonable effort [Yan+12]. As the correlation between temperature and drift is more or less linear, just changing the amount of drift during simulation is probably sufficient to simulate changes in temperature.

3.6.6 Actors

The only actors that are commonly supported, even on real nodes, are LEDs (see fig. 3.5, node 1), which can be found on practically any sensor node for debugging purposes. As these only have two states, simulation is rather trivial. Besides LEDs, actors are too application specific to be found in generic WSN simulators. One such example is mentioned in the next section (section 3.6.7). Operation modes like Pulse Width Modulation (PWM), which are used to dim LEDs are seldom implemented. In this mode the LED is tuned on and off at a frequency that is too high for the human eye to perceive. However, this brings multiple problems, as the simulator must either print the on/off ratio as number or simulate the visual perception, which is not linear. [HS36]

Impact As actors are implementation specific it is impossible to make a general statement. However, as long as there is feedback of the action to the network it is normally safe to ignore the missing actor. For example if the actor switches on a light bulb this has little effect on the network itself, unless the light-bulb heats up the node enough to cause clock-drift.

3.6.7 Sensory

Nodes in a WSN should have some kind of sensory. The commonly used Sky mote has temperature, humidity and light sensors (fig. 3.5). The button for user-input is a sensor, too. Nonetheless, most simulators only provide minimal models for sensory, either by providing a button for digital inputs, or a

slider where the currently read value can be adjusted at run time, or by loading a list of time-value tuples (stimuli). The reason for this is that the required accuracy of the sensory model is highly application dependent and does not only depend on its physical attributes, but mainly on the way the data is processed. For example a pure sensing application that sends data to a sink is agnostic to the actual value it measures and forwards to the sink. It is therefore agnostic to the data it reads. If the data is compressed, though, it can make a big difference if the sensory output is noisy or not. Whether the absolute value is relevant, depends on the algorithm. For a fire detection application, on the other hand, it is only relevant whether the value is above or below a certain threshold.

As a consequence there is no way to provide a generic sensory model. Instead each model must be specifically tailored to the application. To allow the simulation of cyber-physical systems GISOO integrates a WSN simulator and simulator for dynamic systems [Ami+13]. This approach allows nodes to interact with each other, not only via the radio, but also using actors and sensors. For example, if a node detects a low level in a water tank it can request another node to open a valve to refill the tank. Even though, this only provides domain specific tooling support and only simplifies the implementation of the application specific model, but does not make the modeling superfluous.

Impact and Discussion As with actors, the impact of inaccurate or missing support for sensors is very application specific. As long as the code running on the node is only marginally influenced by the data provided by the sensor, this can be tolerated. If the sensory data however influences the WSN, this can become a severe problem. An example for this is data compression: If the data in the simulation does not resemble real data the simulation, results are likely to become over pessimistic or over optimistic.

While raw input data can be fed to the simulation with little effort, interaction between nodes becomes more problematic. In such a case it becomes necessary to model the physical interaction of the nodes and integrate this into the simulator. Unless using a framework like GISOO, which already made the necessary adjustments to a simulator, simulating cyberphysical systems will require not only implementing the physical system, but also a good understanding of the WSN simulators internals to get the timing right.

3.6.8 Network Model

The network is the part of a WSN that is most difficult to model. There are different approaches to tackle the issue – from simple to very complex. In section 2.6 (Low Power Radio Models, page 24) I already presented the different models involved in simulating the network of a WSN. For this reason I will give a brief repetition to discuss the issues of realism.

A more or less perfect mapping of the real world is not only very elaborate, but impossible. Every object and surface dampens and/or reflects the radio signal and changing the orientation of a node or moving it a few cm can change the quality of the connection significantly. For this reason there is a multitude of different radio models. The most complex models require detailed information about the physical environment, enriched with calibration data. [Goe+14] They simulate the actual radio propagation, but also have to simulate the High Frequency (HF) section of the radio transceiver to a certain degree.

More simple models make use of knowledge about the protocol and radio hardware. The modulation technique used and the mandatory Cyclic Redundancy Check (CRC) at the MAC layer make it very unlikely that a transmission error is not detected when using IEEE 802.15.4. Assuming that the micro controller only retrieves successfully received and verified packets from the radio, a simple radio error model that merely distinguishes between successfully and unsuccessfully received packets is sufficient. Provided that the simulation parameters are correctly set to reflect the real network in question, experiments show that it is possible to reproduce results from real networks using rather simple models. [HL10; PPB07]

For less sophisticated radio chips like the also commonly used CC1100, which works at the bit layer, this is not possible. Compared to the IEEE 802.15.4-based chips it uses a less robust modulation technique and only provides a bit stream. The network model must therefore implement random bit errors as well as the results of interference.

Dynamic Environmental Effects Besides the general characteristics, network attributives of a WSN are not only influenced by structural elements, but also by other aspects like temperature, humidity, or interference. The 2.4 GHz band, for example, is shared with 802.11b/g/n (Wireless Local Area Network (WLAN)), Bluetooth (BT) and microwave ovens. [LCL07; Boa+09] This however mainly applies for office environments, where realistic simulations are made even more challenging by people walking around, opening and closing doors, etc. anyway.

Opposed to the expectations induced by poor radio and TV reception during bad weather, humidity seems to impact the low power radios used by WSN nodes less than temperature. [Boa+10a; Mar+13] The latter is important, as temperatures between nodes can differ, depending on whether they are exposed to the sun or lying in the shadow [HIT08]. The actual effect depends on the hardware, the concrete node and whether it is sending or receiving [Boa+10b].

Impact A network model, once developed, can be reused. The configuration must be adjusted for each use case, though. The simulation results only apply to that specific configuration. While it is certainly possible to create a typical scenario for certain use cases, this requires a huge amount of expertise. When simulating one must also be aware of the limitations of the model. Certain effects like asymmetry [Zho+04] or the capture effect (see section 2.6.7.3, page 35) can have a significant impact on the outcome, yet are often not modeled.

Discussion That commonly made assumptions, especially about the network, are not accurate has been researched in detail [Bur+09; Ste+12; YNB11; Kot+04; GJP10; Ber+10]. Even relatively simple models can yield quite accurate results. [HL10; GJP10; Ber+10] Even so, the too simple and therefore not very realistic Unit Disk Graph (UDG) model (see section 2.6.6, page 31) is very popular. The main reason for this is, as is highlighted by many publications about network models, other models must be properly configured. Creating such a configuration out of the blue is not much better than using a UDG model. The effort of creating a good configuration is complemented by the lack of tooling support.

It has also been suggested that it would be a good idea to create a common description language that can be read by different WSN simulators and use it to build a repository of common WSN scenarios [Li+10]. As part of this efforts support for some simulators was developed. Yet these are not maintained anymore and no such repository has been established.

3.6.9 Radio Hardware

The radio hardware is divided in two parts, a digital logic and an analogue HF part. The first can be considered digital hardware (see section 3.6.4, page 57) and can be emulated quite accurately. Only certain operations that have minimum and maximum timings are difficult to model accurately.

The HF part, however, is more complicated. Often the inner workings are not, or only vaguely, explained in the data sheets of the radios. For example the CC2420 datasheet explains the Link Quality Indication (LQI) value as follows:

> This unsigned 7-bit value can be looked upon as a measurement of the “chip error rate,” although CC2420 does not do chip decision. [Tex04]

Similarly the decision whether the channel is clear (Clear Channel Assessment (CCA)) can be configured to be based on the received signal power and the reception of “valid IEEE 802.15.4 data”. It is however unclear whether the preamble of a packet must be received and how many IEEE 802.15.4-chips must be successfully decoded to consider the received signal valid IEEE 802.15.4 data. Creating a model for this, possibly by taking the LQI value into account, requires chip-specific information which is not provided by the freely available documentation.

Impact Most of the missing information is not of major relevance developing a driver for the chip. When trying to understand certain effects or modeling the chip, the lack of knowledge can require further research to better understand the chip.

Discussion It is unclear whether the inaccuracy of the data sheets is laziness, an effort of keeping the documentation simple or an attempt to protect intellectual property. For the modeling of such chips this is most problematic, because these aspects are either implemented as seems most appropriate, which might be wrong, or not at all. Although some aspects have been found experimentally and published, it is rather unlikely that someone who implements a model based on the data sheet does a comprehensive literature search to find additional information. In the publications discussing the issue, I have also found no hints whether the manufacturer has been successfully contacted to resolve such questions. It seems like ignoring such issues for the time being is often the preferred mode of operation.

3.6.10 Energy

Estimating the energy consumption for real WSN nodes is not too complicated. Basically it is sufficient to aggregate the time the components spend in a certain state and multiply that with the energy consumption in that state.

Table 3.2: Energy consumption used by Energest [Dun+07]. The baseline energy consumption of the radio is included in the CPU states.

Component	State	mW
CPU	On	1.8
	Sleep	0.0545
Radio	Rx	20.0
	Tx	17.7
Led	On	4.6

Table 3.2 shows the parameters used for the Sky mote by the software-based energy estimation Energest. [Dun+07] Other nodes have similar values. The CPU states include the energy used by the radio in sleep state and the radio's Rx and Tx values are the offset. This is also the reason why the factor between the active and sleeping CPU is "only" ≈33. Otherwise, it would be much larger. The numbers also show that, against common intuition, receiving data has a higher energy usage than sending. While this does not apply to all low power radios, it is a common energy scheme, especially with those radio chips that have more complex modulations like Offset Quadrature Phase Shift Keying (OQPSK) and Pseudo-random Noise (PN) modulation as used by IEEE 802.15.4 (see section 2.6.2, page 25).

Hurni et al. extend this model for the radio chip by adding constant energy usage for the transition between the states. [Hur+11] The rationale for this is that the transition between states is not instantaneous and uses a different amount of power than any of the other states. For example, when the radio chip switches from Rx to Tx, the receiver is turned off and only after a few ms the sending unit is turned on. During this time the energy usage drops to a significantly lower level. When adjusting the factors to that of a certain node, Hurni et al. managed to decrease the deviation between nodes and simulation to almost 1 %, which is lower than the production-caused deviation between the nodes used for their experiments.

Haas et al. took a similar approach for the instruction level simulator Avrora, but also included the CPU's states [HWS12]. Again their accuracy was higher than the deviation between their nodes.

Discussion As shown, modeling the energy consumption using a simulator can be done with an accuracy that is higher than the variances between devices of the same type. This accuracy can easily result in the assumption that the energy estimation of the simulator is accurate, which is correct for the simulated node, but not the simulation. Clock drift is normally not modeled by simulators, yet the loss of synchronization between nodes can require them to send more packets, which in turn has a considerable impact on the energy consumption. The same applies for any other inaccuracy in the simulators model.

The example shows that the impact of a model's inaccuracy strongly depends on the software running on the node. Temperature sensors for consumer weather stations typically do not have a radio receiver at all. They just broadcast the values at a regular interval with a random offset to avoid collision. Their energy consumption can be accurately calculated, as they are neither influenced by the network model, sensor reading nor clock drift. As there is no interaction with other nodes and a single period can be interpolated to the node's lifetime, it is probably easier to use real hardware, an oscilloscope, pen and paper.

Multiple models have been presented to predict the lifetime of a node more accurately. [Per+08; Ker+10; Spo+08] Yet all these ignore the discussed elements of gaining accurate information about the actual energy consumption of the different components. I therefore doubt that the presented accuracy can be reproduced outside the benchmarks used for the evaluation. Modeling a battery becomes even more questionable as the model must be calibrated to each battery model and brand and sways the developer in false accuracy.

3.6.11 Discussion

The analysis shows that there are three main aspects of WSN simulation. First there are those parts that can be accurately modeled or emulated. This applies to most of the hardware, specifically the micro controller and the digital components of the radio. The second component is the radio. It can only be simulated. There are multiple models, from simple to complex. While even relatively simple models yield good results, the authors of those publications always stress that these results require proper tuning. The third component are sensors and actors that interact with the outside world. These are always application specific and must be modeled for each use case.

The digital part of the system has two main sources of unrealism. The first are bugs. They can source in implementation mistakes, misinterpretation of or errors in the reference (e.g. data sheet), or the lack of information. The second type are missing functionality. Due to the mechanisms used of the software interacting with the different function units of the micro controller, the absence of a certain functionality is silently ignored.

Tracking down missing features and bugs can become tedious. Typically, they do not cause a component to fail completely, but it just does not behave as expected. As the rest of the simulator works accurately, it is easy to suspect the cause of the misbehavior in the software being simulated. This is however doomed to fail, yet costs a lot of time.

While the number of different WSN nodes available is limited, every WSN deployment is different. The only way to model WSN networks more accurately is to either provide a repository with reference environments that can be loaded into the simulator, or to provide tools to simplify the configuration. This can either be done by using a generator, which will require a lot of additional research, or by taking a real network as reference, as done in this work.

Missing features or inaccurate network models and configurations can cause unpleasant surprises, when deploying the software on real nodes. Most notably these are effects like clock drift, and fluctuations and asymmetry in the connections.

For the user of the simulator the question arises how to deal with effects not reflected by teh simulator. Stojmenovic [Sto08] makes a very strong case that one should use the simplest model possible and increase the complexity and accuracy during the development. The reasoning behind this is that if something performs bad using a simple and therefore idealistic environment, it is much easier to debug without having to cope with complex effects.

However, when not being sufficiently aware of the limitations, this approach can require the redesign of components. In their experience report Stecklina et al. [SK13] describe their experiences using a NLS during the development. The transition to real hardware was made late during the project, causing multiple pitfalls. They specifically note issues like unaligned memory access, stack size, interrupt handling, runtime constraints and compiler bugs. In consequence they suggest to use ILS more extensively.

Yet there are limitations even to emulating the hardware as far as possible and simulating the network accurately. Some of them are timing related. Often data sheets only provide lower and upper bounds for certain operations or explain theses only very vaguely (see section 3.6.9, page 61). Without background knowledge finding the exact behavior experimentally is impossible, too, as there are too many attributes that might influence the result. It is only possible to find a likely behavior that applies to the environment provided by the test cases used to examine the concrete behavior. It must be verified that it also applies to the environment provided by the final deployment.

In the end simulation results must always be verified in the real world if robust results are needed. Sometimes this is not possible, though. For academic research there is no suitable testbed available to test a several 100 node deployment, although this can be necessary to evaluate a network protocol. Even if a model has been very well tested, the simulation results must be treated with caution.

3.7 Analysis of the Background Noise

The background noise can strongly influence the connection properties between nodes. Techniques like CSMA try to reduce the impact of other nodes sending, as well as external noise. The characteristics of external noise sources have already been discussed in section 2.6.4, page 29. In the evaluation I will also show that the impact of the noise differs between protocols.

To get a better understanding of the background noise that can be found in the testbed in Erlangen, I ran a one week survey. In this section I will present the application I developed to sample the noise at a high as possible rate. I will then analyze the data and discuss how this impacts simulation models.

3.7.1 Monitoring Application

To monitor the background noise, the monitoring application has to read the RSSI radio via the Serial Peripheral Interface (SPI) bus and then transmit it via the serial. If possible, this should happen at a sample rate that is close to the sample rate of the radio. The CC2420 radio chip, which is used on the target node, averages the incoming signal strength over 128 µs. Thus an optimal sample rate would be higher than 8192 Hz to meet the Nyquist–Shannon sampling theorem.

It was not possible to achieve this using ContikiOS, because the abstraction layer introduced too much overhead. For one, the interrupt handling took more time than the sampling itself. Also, the time-keeping

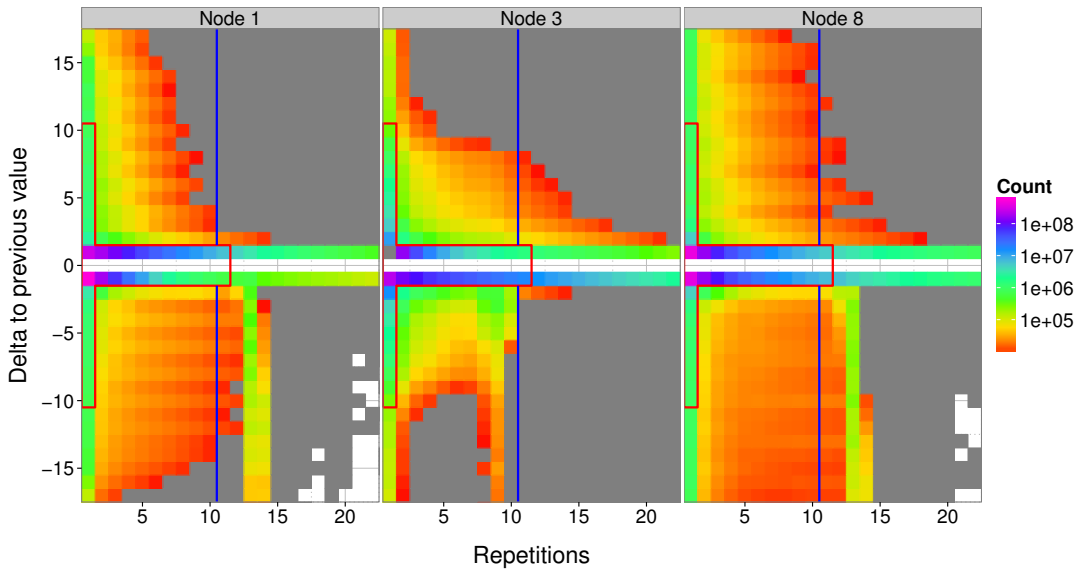


Figure 3.6: Repetitions of a RSSI value based on its delta to the previous value. The gray areas contain values that are lower than the scale, which starts at 10^4. The white area contains no data. The red line surrounds the data that can be encoded using one byte. The data to the left of the blue line is encoded using two bytes, the data to the right using four bytes. The data is based on 1.6×10^9 samples

interrupt introduced a strong jitter. Additionally, the abstraction layer for the radio hardware had multiple sanity checks to ensure that the radio was in the expected state and handle exceptions. This introduced a unnecessary overhead for this specific use case.

To increase the sample rate, the Contiki OS was only used to initialize the system, but not to collect the data. After starting the thread that sampled the data, all interrupts were masked and were not handled any more. Instead the thread actively waited for the timer interrupt flag to be set and also cleared it. Care was taken that the duration of the operations done between two checks were short enough not to miss any interrupts.

The second optimization, was to minimize the overhead when communicating with the radio. As the radio is in a well known state while sampling, it was possible to remove unnecessary checks and communication with the radio. Additionally, all functions required while sampling were migrated to the same compilation unit to provide better optimization possibilities to the compiler. With these optimizations it was possible to achieve a reliable sample rate of 8192 samples/s.

The application does not only have to collect the data from the radio, but also publish it via the serial. The 8192 samples per second can theoretically be transmitted using the serial interface with 115 200 baud, allowing to transmit ≈11 520 B/s, which is sufficient to transmit the raw data.

To use this in the context of the Wisebed testbed management software it was desirable to use a 7 bit ASCII encoding to avoid issues caused by conversion within the software stack. The bandwidth is however not sufficient to map each byte to two characters.

The analysis of the collected data has shown that if the change to the previous reading was small, the next reading was likely to be the same as the current. If however the change to the previous reading was large, it was very likely that the following change was large, too. Figure 3.6 visualizes this for three different nodes. The numbers will be discussed in section 3.7.2.

Based on these numbers an encoding was used that first counts the number of consecutive readings (run length) and encodes them based on three different classes. The classes are visualized in table 3.3 and also marked in fig. 3.6. Following the nomenclature of other encodings I will call one encoded data set a chip.

The first class encodes delta/repetition tuples that are either single values that differ less than 10 from the previous value or that are off by one and have a run length less than 12. These are mapped to subsets of 40 characters of the ACSII table. This choice was made to be able to transmit as many of the possible tuples as possible using a single character. The preference was given to the delta over the run length, as a high run length also provides more time to transmit the data. In fig. 3.6 this class is marked with a red line and in table 3.3 with a reddish background.

Table 3.3: Excerpt from the encoding table used to encode the delta/run-length tuples.

Delta to previous value	Repetitions											
	1	2	3	4	5	6	7	8	9	10	11	12
-45	–	–	–	–	–	–	–	–	–	–	–	–
-44	i!	j!	k!	l!	m!	n!	o!	p!	q!	r!	–	–
⋮												
-11	iB	jB	kB	lB	mB	nB	oB	pB	qB	rB	–	–
-10	J	jC	kC	lC	mC	nC	oC	pC	qC	rC	–	–
-9	I	jD	kD	lD	mD	nD	oD	pD	qD	rD	–	–
-8	H	jE	kE	lE	mE	nE	oE	pE	qE	rE	–	–
-7	G	jF	kF	lF	mF	nF	oF	pF	qF	rF	–	–
-6	F	jG	kG	lG	mG	nG	oG	pG	qG	rG	–	–
-5	E	jH	kH	lH	mH	nH	oH	pH	qH	rH	–	–
-4	D	jI	kI	lI	mI	nI	oI	pI	qI	rI	–	–
-3	C	jJ	kJ	lJ	mJ	nJ	oJ	pJ	qJ	rJ	–	–
-2	B	jK	kK	lK	mK	nK	oK	pK	qK	rK	–	–
-1	A	U	V	W	X	Y	Z	[	\	]	^	–
1	K	_	‘	a	b	c	d	e	f	g	h	–
2	L	jO	kO	lO	mO	nO	oO	pO	qO	rO	–	–
3	M	jP	kP	lP	mP	nP	oP	pP	qP	rP	–	–
4	N	jQ	kQ	lQ	mQ	nQ	oQ	pQ	qQ	rQ	–	–
5	O	jR	kR	lR	mR	nR	oR	pR	qR	rR	–	–
6	P	jS	kS	lS	mS	nS	oS	pS	qS	rS	–	–
7	Q	jT	kT	lT	mT	nT	oT	pT	qT	rT	–	–
8	R	jU	kU	lU	mU	nU	oU	pU	qU	rU	–	–
9	S	jV	kV	lV	mV	nV	oV	pV	qV	rV	–	–
10	T	jW	kW	lW	mW	nW	oW	pW	qW	rW	–	–
11	iX	jX	kX	lX	mX	nX	oX	pX	qX	rX	–	–
⋮												
44	iy	jy	ky	ly	my	ny	oy	py	qy	ry	–	–
45	–	–	–	–	–	–	–	–	–	–	–	–

The second class are changes of less 45 and less than 11 repetitions and are not covered by the first class. Run lengths of this class are encoded using 2-byte-chips, where the first character defines the length, and the second the offset to the previous value. While the first character does not overlap with the other classes, the full ASCII range is used for the second character. The data that can be encoded using this class is to the left of the blue line in fig. 3.6 (Repeated RSSI values vs delta, page 64) and is highlighted in blue in table 3.3.

All other combinations fall into the third class, which are encoded using classical run length encoding, using two characters for the hex-encoded value and repetitions each, resulting in 4-byte-chips. The values used do not overlap with the other classes, except for the second value of the second class.

Depending on the number of repetitions of a value, transmitting the data can take longer than the time to sample it. To compensate for those that take longer to transmit than to sample, a buffer was used. If the buffer is full, the data is sampled to a special value. This way, while the actual data is lost, the data loss does not stay undetected. Due to the run length encoding, the special value can be easily sampled long enough until the buffer is sufficiently emptied.

To ensure that the program works as expected, 40 chips are followed by three lower hex-encoded nibbles of the sample counter and a newline character. This technique allows to verify that no characters or lines were lost, as the number of the decoded samples and the counter would otherwise differ.

Discussion Using the delta-encoding does have a big disadvantage: To start decoding the data a reference value is required, which can only be acquired when the data falls in to the third class, which provides an absolute value and requires four bytes to encode the data. Experience has shown that the chance of this happening is sufficiently high. Due to the high sample-rate (8192 Hz) having to discard the first several thousand samples can easily be tolerated.

Table 3.4: Repetitions of a RSSI value based on its delta to the previous value. The data is based on 1.6×10^9 chips

	Node 1	Node 3	Node 8
Encoded using 1 byte	96.88 %	92.36 %	93.06 %
Encoded at least 2 samples with 1 byte	48.00 %	43.06 %	45.29 %
Encoded using 4 bytes	1.22 %	5.96 %	2.20 %
Average number of samples per chip	2.53	3.43	3.16

The second issue with this approach is that the internal buffer must be large enough to compensate for data from class two and three. This is basically a statistical matter, and there is no guarantee that all data can be captured. This is especially the case in a very noisy environment where the delta is too large to be encoded using a single byte encoding and where the run length is 1. In this case writing the data to the serial takes more than four times longer than to sample the value. Even though, after sampling more than 7×10^9 readings, most nodes did not miss a single sample. Even sampling at 10 923 Hz worked quite reliable for selected nodes. Sampling at rates above 10 923 Hz was limited by the SPI connection to the radio chip.

Table 3.4 shows some statistics about the data collected by the nodes 1, 3 and 8. The majority of the data (>90 %) was encoded using a single byte. More importantly, about every other 1-byte-chip encoded two or more samples using a single byte. The fact that data was encoded using the third class, requiring 4 bytes is not necessarily a bad thing, as run lengths greater 10 respectively 11 also fall into this category.

3.7.2 Analysis

Using the monitoring application the network was surveyed for about a week. The data was made available via Zenodo [Str17b]. From this data set 5×10^9 samples of nodes 1, 3 and 8 in Erlangen ware taken and statistics calculated how often the same value was read repeatedly from the radio chip. Figure 3.7 presents the worked up data. The measured signal strength in $\approx$ dBm lies on the Y-axis, while the X-axis represents the run length. Values for the signal strength are only approximations, as accurate data requires a calibration using the antenna. However, the numbers are proportional to the signal strength that arrives at the radio chip, which actually is of higher relevance. The combinations of measured value and its repetitions is counted and visualized by coloring it using a logarithmic scale to the base of 10.

All nodes show a base level, marked with **1**, around −100 to −90 dBm which is probably due to thermal noise. That the radio chip returns values as much as 20 dB below this value cannot be explained with noise. It will be picked up in the discussion together with even values being sampled more often.

Far more interesting are the attributes that can be seen for higher signal strength. There is a clearly visible area of measurements of discrete length marked with **2**. These values are clearly caused by traffic by other radios like BT or WLAN, yet most likely the latter. They have in common that they end with a darker value. This means that these run lengths were encountered more often than the shorter ones.

Interestingly the duration for node 3 is shorter than that of nodes 1 and 8. A possible explanation could be that node 3 was placed near an WLAN access point, and the typical packet length differs. A timing-problem can be ruled out, as the number of samples collected from the node during the experiment is in accordance with the other nodes. Nodes 1 and 8 have high numbers of run lengths of up to 14, while for node 3 the zone of a high number of repetitions ends at 10. Yet while the other two nodes seldom have more than 40 repetitions above 90 dBm, node 3 has quite a few of them.

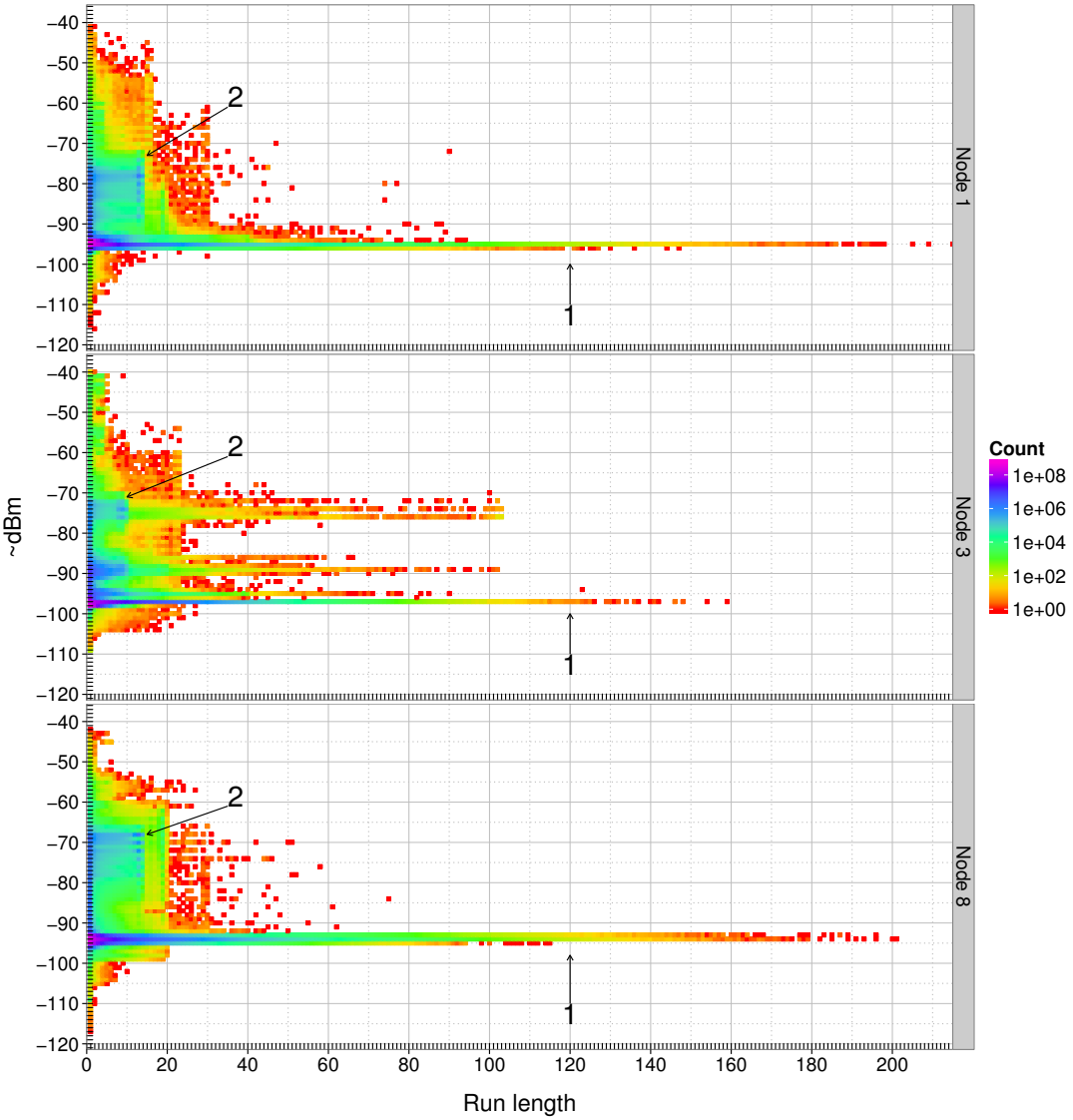
Its also visible that the signal strengths differ. While node 3 seems to be interfered by two discrete sources, one at about −75 dBm and one at −88 dBm, the other two are interfered by a single source, or multiple that are received at a similar signal strength.

As the signal seems to have a maximum length, the lower repetitions might be caused by jitter in the signal strength. Meaning that instead of receiving a signal with −80 dBm and a run length of 13, the data contains a signal of −80 dBm and a run length of 7 as well as a signal of −79 dBm and a run length of 6.

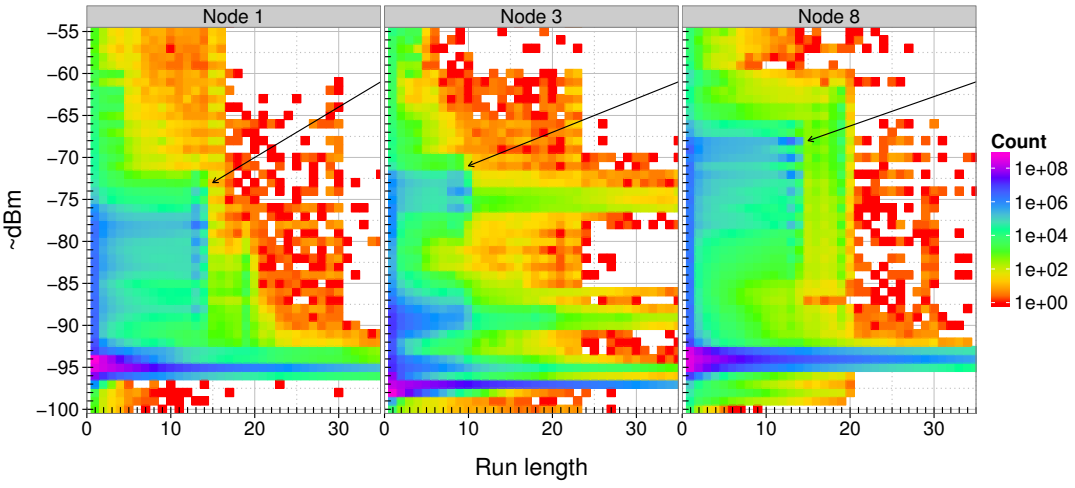
Another interesting effect is that even values are more likely to be sampled than uneven ones. It is especially visible in the area marked with **2** for nodes 1 and 8, and to the right of this area for node 3. As the data is not preprocessed, and the effect can be seen for all nodes this must be caused by the hardware. Possible causes might be the ADC that converts the signal strength, or the averaging algorithm.

To check this theory the data was filtered. The data was aggregated to a single run length under the following conditions:

- The sample of the signal strength does not differ more than 3 dBm from the previous value.

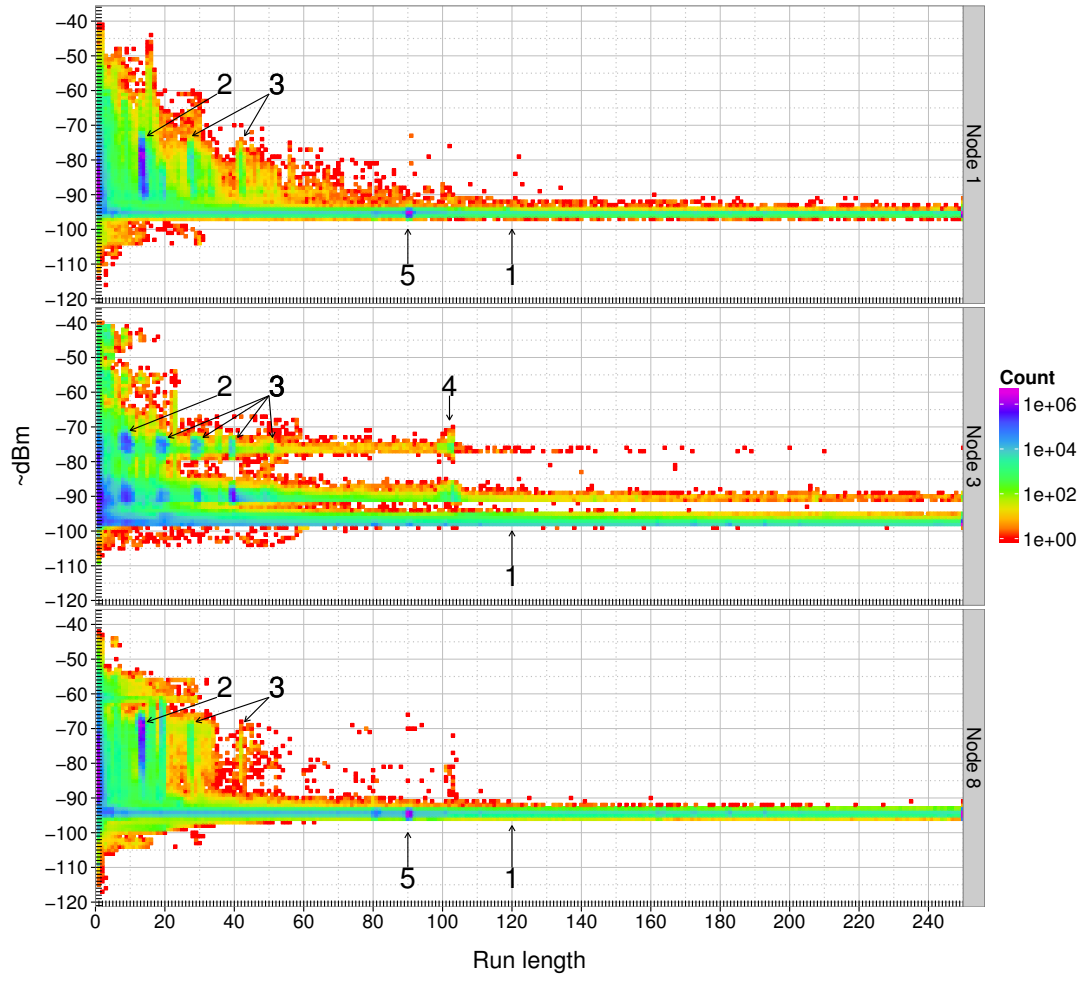


a) Whole data range

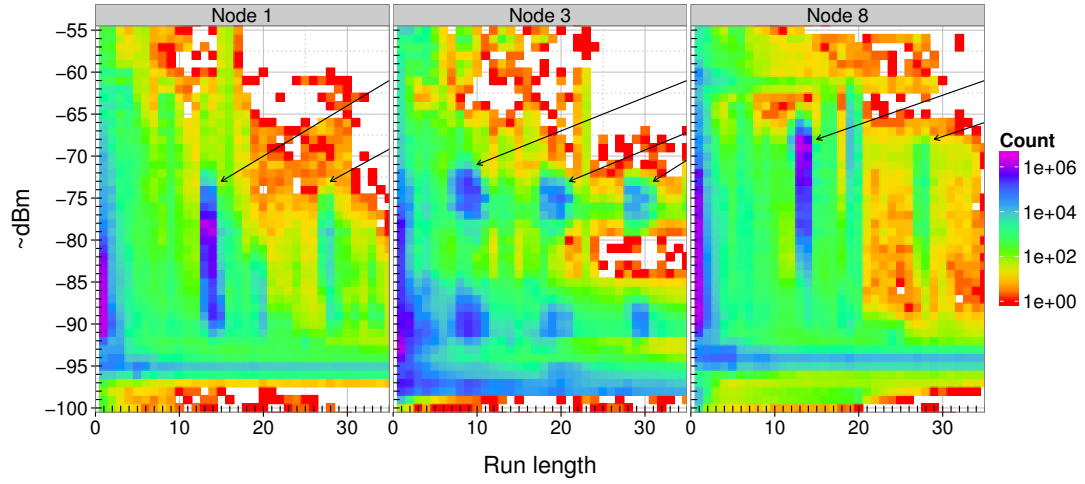


b) Zoomed area of interest

Figure 3.7: Received signal strength vs duration. The data was sampled at $8192\,\mathrm{Hz}$. The dataset contains 5×10^9 samples. Run lengths above 250 are binned to 250. Data: [Str17b]



a) Whole data range



b) Zoomed area of interest

Figure 3.8: Contains the same data as fig. 3.7 (Background noise: Signal strength vs duration, page 67), yet the data was filtered to allow longer run lengths if the signal only changes slightly. Run lengths above 250 are binned to 250. Data: [Str17b]

- Tthe maximum and minimum of the aggregated data does not differ more than 5 dBm.

The results are shown in fig. 3.8.

The filtered data does show some interesting effects. The fixed durations that can already be seen in fig. 3.7 and marked with **2** are even more visible. For nodes 1 and 8 the area is distinct in repetitions, covers over 10 dBm and the count goes up to well over 1×10^6. Node 3's area however is rather circular, and the count stays at 1×10^4 to 1×10^5. Although, due the scale, it is not immediately obvious, node 3 receives fewer strong signals than the other two nodes.

In addition to the duration marked with **2** becoming more visible, multiples of this duration become visible and are marked with **3**. The most likely explanation for this effect are multiple back-to-back transmissions: The average RSSI value calculated by the hardware drops, yet only one or two dB. Depending on when the data is sampled this is sometimes enough to start a new tuple and sometimes it is not.

Node 3 shows additional effect. Firstly, four instead of only two multiples of the run length marked with **2**, which are marked with **3**, become visible. Secondly a run length of about 103 was captured quite often and is marked with **4**; This is about 10 times the run length marked with **2**. The similarity of the run lengths captured at −75 dBm and −90 dBm might hint at them being caused by the same source, yet under different conditions (e.g. close/open door).

Nodes 1 and 8 seem to have a quite common duration of silence at 90 repetitions, which is marked with **5**. A possible explanation for this might be packets that are sent back-to-back with a short break in between. It could also be that this is a reply sent by another radio. To answer this definitively more research is required, probably of the WLAN standards, which goes beyond the scope of this work.

The data presented is based on 5×10^9 samples, an amount of data that cannot be processed on a node. It might however be possible to aggregate subsets. To check whether this might be possible, the first 10 000 samples of every 10 000 000 samples provided by node 8 were filtered using the aggregation algorithm just presented. Each of this sub sample set is shown as separate graph in fig. 3.9.

Most graphs show a well visible cluster at about 20 repetitions. Its signal strength, however, does differ significantly. These are clearly caused by some other device transmitting data.

Besides these clusters most samples show tuples representing either long durations ($\geq$50) of silence at under −95 dBm or single samples of higher signal strengths. Samples 1, 2, 9 and 55 as well as 61 to 65 are an exception. They also have other tuples: Specifically stronger signals with a run length of two or three and shorter durations of silence. While an increase in traffic might be the cause for the shorter silence, this is rather unlikely, because the number of samples in the cluster is often below average. More likely there was additional noise, possibly caused by cross-channel interference that caused the signal strength to change sufficiently enough to start the next signal-strength/run-length tuple. To find the cause of this effect further investigations are required.

Additionally, statistics were generated about possible packets. All data that had a signal strength of ≥ -90 dBm and $\geq$5 repetitions were considered a packet. In fig. 3.9 and fig. 3.10 the tuples are separated by a red line. The graphs show this information as "tup", which shows how many of the tuples fall into this category. "smp" gives the amount of samples represented by these tuples. Depending on the sample set, about 25 to 30 % of the collected tuples fall into this category (Min: 11.4 % Mean: 27.8 % Max: 35.7 %). Yet they only present about 4 % of the samples (Min: 3.1 % Mean: 3.7 % Max: 5.8 %). Thus, the average tuple does not fall into this category and has a much higher run length.

As already visible in figs. 3.7 and 3.8 the data collected by node 3 looks different. There many more tuples receiving a high signal with many repetitions. The node also has less signal activity, although this is not as visible due to the logarithmic scale used in the graphs. Figure 3.10 shows the same data as before, but for node 3 instead of node 8. The most obvious change is that many of the sample sets do not contain tuples that represent packets at all. Specifically the sample set 15 contains a single tuple with a run length of 10 000 and sample set 31 contain two tuples with the same value. The latter is caused by the algorithm starting a new samples set, not only when two concurrent samples differ more than 3 dB, but also when the current sample differs more than 5 dB from the extrema added to the current tuple. Sample sets 39, 43, 50, 51 and 72 are similar.

Even though there are sample sets that do not have any tuples that are considered a packet, the overall averages do not differ too much from the sample set provided by node 8. The average number of tuples being considered a packet is even slightly higher at 28 % (Min: 0.0 % Mean: 28.3 % Max: 48.7 %). These numbers are however a bit misleading. If a set consists of one tuple being considered a packet and one tuple of very long silence, 50 % of the tuples fall into the category of packets. Thus, the number of packets represented by these tuples are is meaningful. Interestingly the mean number of packets is even higher than those of node 8 (Min: 0.0 % Mean: 4.1 % Max: 5.9 %). In conclusion node 3 has longer periods of silence, but it also has more active durations than node 8.

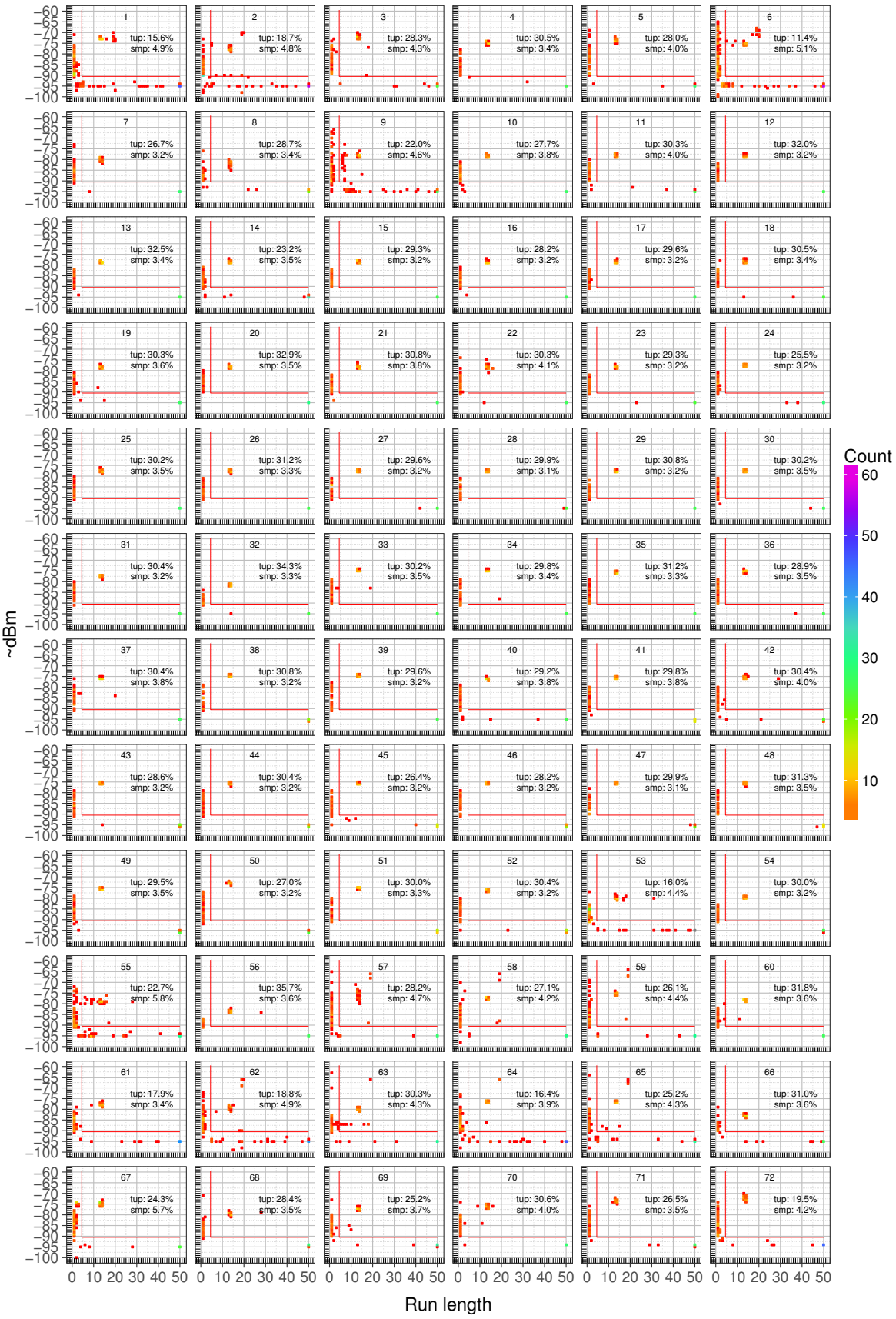


Figure 3.9: Each graph shows the first $10\,000$ samples of every $10\,000\,000$ provided by node 8 as used for fig. 3.7. The data is caped to the limits of the graph. Thus signal strengths are above $-60\,\mathrm{dBm}$ and below $-100\,\mathrm{dBm}$ are set to those values. Accordingly run lengths above 50 are capped to 50. **tup** shows how many of the tuples represent a signal strength higher than $88\,\mathrm{dBm}$ and more than 9 repetitions. The tuples that fall into this category are separated by a red line. **smp** gives the number of samples represented by these tuples. Data: [Str17b]

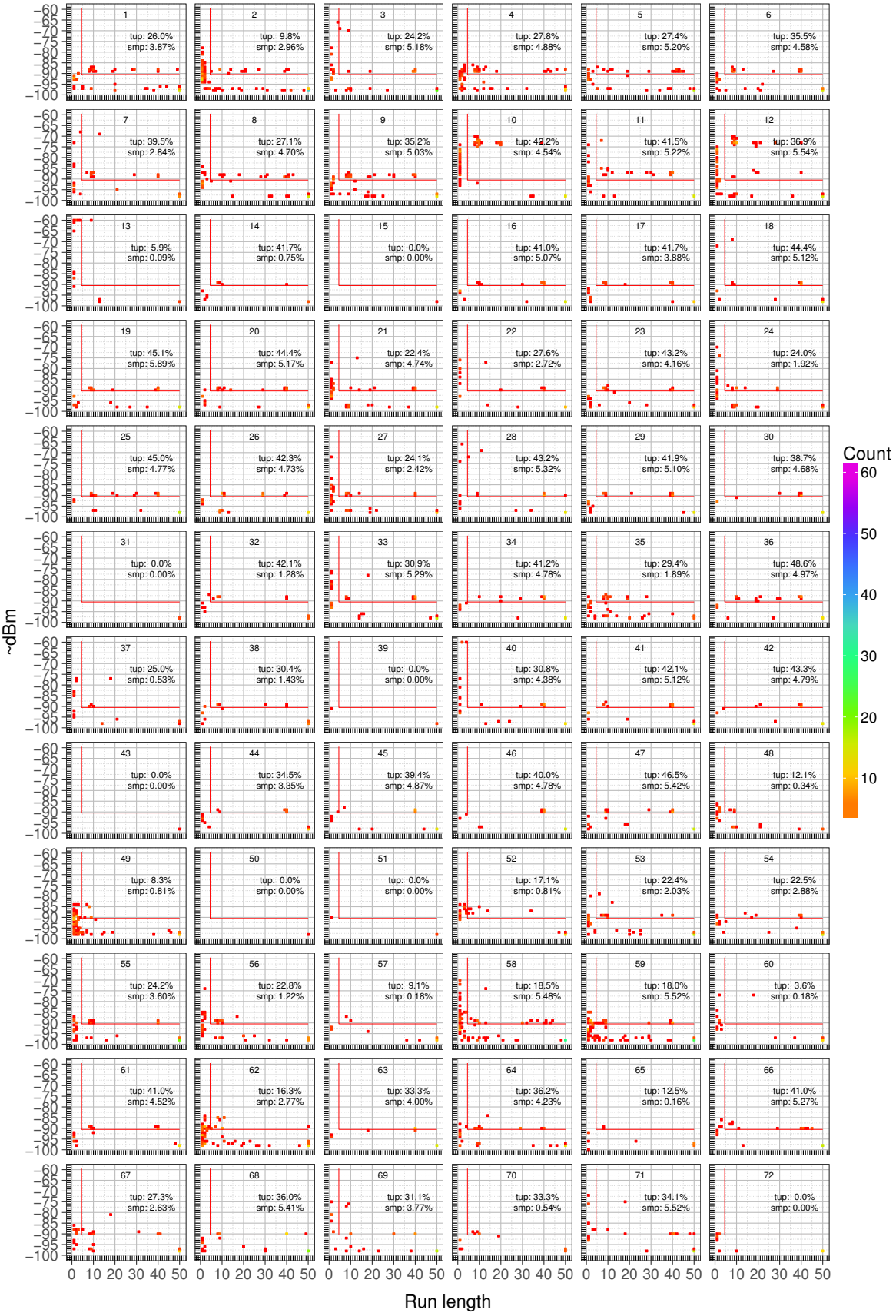


Figure 3.10: The graphs show the same data as in fig. 3.9, yet for node 3. Data: [Str17b]

3.7.3 Discussion

The data presented was collected using a special firmware that uses the complete bandwidth of the serial interface. Sending the same amount of data using a wireless connection is not possible. As Palm showed [Pal12] the net bandwidth of an IEEE 802.15.4 radio is sufficient to transmit the data. Yet this only applies to a single link and in optimal conditions. Receiving or forwarding is not possible, at least without a second radio using another channel. Besides, while sending data, the RSSI cannot be acquired. Thus the only solution is to buffer the data on the node itself, which requires more resources than commonly available. The filtered and aggregated data presented in figs. 3.9 and 3.10, however, reduced the amount of data significantly, while the algorithm is simple enough to be run on a micro controller efficiently. Yet is the reduction sufficient?

The number of tuples created by the data analyzed data sets of 10 000 samples range from 1 to 125. However, the third quintile for both, nodes 3 and 8, is below 45. Additional analyses can likely come up with a filter that is able to further reduce the number of tuples without the loss of significant information. The signal strength can be saved in a single byte, as the raw value is 7 bit. As the number of samples is limited, a 16 bit value is sufficient to save a run length up to 65 536. How often a tuple is counted is a statistical matter, however. For the 72 sample sets of 10 000 samples each, the maximum for the two nodes was 24, and therefore using a 8 bit value most likely is sufficient. Concluding this requires 4 bytes per tuple. Assuming that the number of distinct tuples can be kept below 50 – this includes over 3/4 of the presented sample sets – this would require a total of 200 bytes. While this is a lot for a WSN micro controller, it is still an amount that can be provided. This approach therefore seems promising to acquire snap-shots of the network-characteristics. Continuous monitoring is, at least for the class of nodes investigated, not possible, because it is not feasible to do much other processing when sampling at this hight rate.

Unfortunately all this information is of little worth without a model. Lee et al. [LCL07] provide some ideas on how such a model could look like. Yet further research is required to achieve sufficiently accurate models for urban environments. For one their sample rate was at 1 kHz while the data presented was sampled at about 8 kHz and therefore has 8 times the resolution. Considering that many clusters have a width of only two or three samples, sampling at a rate of only 1 kHz is clearly too low. Actually, sampling at 8 kHz, and therefore with about the same period as the width of the low-pass filter provided by the hardware, is too low to meet the Nyquist–Shannon sampling theorem. Yet with the hardware used, it is neither possible to increase he number of data points used by the averaging algorithm, nor the samples rate.

In conclusion more research is required to create a model that allows collecting the required parameters on the nodes themselves. For this reason I will not further look into the data, but will look into whether it is possible to yield reasonable simulation results without a sophisticated noise model.

Validity of the RSSI Values Great care was taken that the data was not altered during different steps. For the node this was done by aggregating the data (e.g. calculating the sum) and verifying that it matches with the decoded data. The data analysis was verified by looking at corner cases and cross checking them against the raw data.

The radio chip however could not be verified due to the lack of hardware tools and limited scope of this work. Nonetheless, it should be noted that the radio chip did return values that could not be explained. Specifically RSSI values that were below an equivalent of -105 dBm. Already values of -100 dBm are rather uncommon. Depending on the node 5.2×10^{-9} to 1.3×10^{-5} of the 5×10^{9} samples had values below -105 dBm. These showed up out of the ordinary and could not be explained with a strong drop in signal strength or similar. Speculating, they might be caused by a concurrency issue in the hardware of the radio chip. If the other values are within the normal range, this can also explain the many tuples with a run length of 1 that can be seen in fig. 3.9 and fig. 3.10.

Analyzing the data also show that even values were sampled more often than uneven. This is not only visible in the raw data fig. 3.7, but also in the delta-encoded data of fig. 3.6. Although most changes are those by one, greater deltas are more likely to be even than odd. Again it can only be speculated. Either the division of the averaging algorithm or the ADC used to determine the input power is biased. If the latter is the case, then this is more likely a design flaw than a random effect, as it can be seen on all nodes analyzed.

3.8 Suggested Approach: Trace-Based Mapping to Enable Simulation-Supported Optimization

To support optimization based on simulation it is vital that the simulation is realistic. It must also resemble the target environment. Meaning, hardware and network properties must be reflected by the simulation.

The analysis shows that the two components of a WSN, the hardware and the network, have been addressed and accurate models have been developed for both. It however also shows that systems that go beyond micro-benchmarks can only be evaluated if the hardware is emulated and the configuration of the radio network is based on a real network. To the best of my knowledge no holistic approach has been researched. Instead, these two aspects have only been analyzed individually, or with a strong focus on one of them.

To achieve accurate hardware simulation an ILS must be used. For the network rather simple models seem sufficient. However, these must reflect realistic properties. This can be a challenge, because predicting real-life properties is almost impossible and they also change over time. To provide a realistic enough model to enable simulation-supported optimization, I suggest using an ILS based WSN simulator and extending its network model to support replaying traces of a real network. Even though using traces only reflects the past and can make no predictions of the future, it is assumed that the traces from the past are closer to the future of the network than any manually generated configuration.

3.8.1 Related Work

Although there has been no real holistic approach before, there has been some work that points in a similar direction. Multiple approaches are based on NLSs like NS2 or OMNeT++ [HB09; GJP10; Mös+12; JR13; Gar+11; GCM11], the WSN Simulator Castalia [Ber+10; PPB07; KVV11] or other not specifically WSN based simulators [Mar+10]. All of them lack an ILS, which can provide accurate runtime timing information.

Some of these approaches try to limit the effect of not having an ILS, for example by adding empirical delays at different layers (e.g. [JR13; GCM11]). These however only apply to their specific benchmark. Changes in code or tool chain can render them obsolete.

Another suggestion is to extend a NLS using a SLS like TOSSIM (e.g. [Gar+11]). This does add some realism, but is still not timing accurate, or rather only in so far as supported by the SLS. Additionally, the OS must be supported by the SLS (see section 2.4.3, page 21).

Instead of tracing the network to configure the network model, simulation has been suggested to gain the necessary data. This is likely possible for areas with little human interaction [AKO14; Ote+14; Hab+13]. He et al. use simulation to predict the PRR for the connections in an office environment before the deployment. [He+12] Considering the strong fluctuations and the effects I saw when monitoring the testbeds, specifically the background noise, I do not think it possible to get reliable results without actual measurements. Their approach seems promising to gain a good initial setup, though.

A problem with traces is the large amount of data created. A solution to this is to generate a model based on the trace and use this for the simulation. Especially approaches using Markov model seem promising. [KCC13]

Instead of making the simulation more realistic, it is also possible to generate an environment for the real hardware. WiNeTestEr [Sub+14] is a channel emulator. Using cables, the antennas of the radio chips are connected to special hardware, which can control the connection properties between the nodes. Opposed to simulation, this approach is not prone to problems in the model or bugs in its implementation. It is however limited to running a single experiment at a time and cannot provide the debugging capabilities provided by simulation. Due to its controllable environment it is suitable to be used as reference to verify a simulator.

Other approaches try to use expert knowledge instead of simulation to optimize the WSN. pTunes [Zim+12], for example, continuously optimizes the MAC layer. While these approaches can probably yield better results than the generic approaches, they can only tune the use cases they are specifically designed for. In the case of pTunes only two specific MAC protocols are supported. When using other protocols, support for them must be implemented in pTunes.

3.8.2 Mapping Networks to Simulators

When mapping real networks to simulators there are limitations. They are imposed by both partners – real network and simulator. The WSN hardware and its available resources limit what and how accurate data can be collected, while the simulator limits the information it can use to influence the simulation.

As discussed in section 3.6, page 52, there are models to derive other parameters like RSSI and PRR from the distance between two nodes and therefore their location. While these models can give a good estimation under certain circumstances, too many factors influence the actual network to give accurate predictions. Even complex commercial models require information about the physical structure and reference measurements to calibrate the model [Goe+14]. Therefore, if actual measurements of the connection properties are available, it is much more reasonable to use them instead of estimations. This does not mean that the additional information, like the position of the nodes, should not be used to enrich the simulation, if only to improve the visualization.

The minimal functionality a transceiver must provide is sending and receiving data. By sending probing data with ids it is possible to create a directed graph with the PRR – or in case of streams the Bit Error Rate (BER). The graph can be enriched with other properties provided by the radio chip. These can either be evaluated by the radio model, for example to support simulating the capture effect(section 2.6.7.3, page 35), or just be provided as value to be read from the radio chip.

Resolution and Resources To model certain attributes, it is important to have precise data of the environment to be modeled. Yet the resolution of the data that can be acquired using the nodes is strongly limited by the available resources. In section 3.7, page 63 this has been discussed for the acquisition of the background noise.

The same applies to all other attributes. The required resolution depends on the monitored property, the environment and the usage scheme. For example, if the PRR is always 100 %, the sampling resolution has no influence on the result. If there is a clear pattern to changes of the value, a higher sample rate might be able to reflect this and allow for more realistic simulation. Yet if packets are sent randomly with no timely correlation, the impact on the overall result might be minimal.

Derived and/or Correlated Information The CCA reflects the radio's decision whether the channel is clear. Thus it correlates with what es happening on the configured channel. It is therefore not reasonable to sample the CCA, but to derive it from the channel properties by the simulation model.

In this case the correlation is clear. More problematic are correlations that are hard to detect automatically, for example when two nodes have very similar connection properties, because they are close together. In a very noise environment those nodes can achieve a very high PRR using CSMA while having high packet losses without, as they delay sending the data to periods with low noise. The next section will also take a closer look on how CSMA affects the collection of PRR information.

As preliminary experiments showed other issues shadowed the possible effects of low resolution data and correlations. They will therefore be neglected for the time being, yet should be kept in mind as possible cause for deviations between simulation and real world.

3.8.3 Collecting Network Attributes and Their Limitations

As the limitations are strongly induced by the hardware I will once again focus on the IEEE 802.15.4-based CC2420 radio chip. Most of the conclusion a generic, or at least apply to most IEEE 802.15.4-based systems.

When collecting data from a network, the data applies to this specific network and nodes. As all antennas are directional, the data even only applies to that specific orientation of the nodes. Then again, other environmental changes that are not under ones control are likely to have a higher impact than the exact position and orientation of the nodes. If the connectivity of two nodes is in the transitional zone, where nodes are just connected (see section 2.1.2, page 16), this might make the difference.

3.8.3.1 Packet Reception Ratio (PRR)

The PRR reflects the ratio of sent packets that are received by a node. It can be acquired by sending enumerated packages. The receiving node can then detect how many of the sent packets got lost. While this seems rather trivial, there are some aspects that must be taken into consideration.

Gaussian Packet Loss Simulators model the PRR with the assumption that it is Gaussian, meaning that the packet is dropped based on a random number. Causes of non-Gaussian packet loss are discussed in section 3.6.2, page 54. Taking these into account requires a PRR model that goes beyond a simple

PRR based probability. The parameters of the model must then be acquired using some model-specific algorithm.

Collisions Opposed to external effects causing packet loss, collisions within the network can and must be handled by the simulation model. Otherwise the correlation that a packets gets lost when two nodes are sending at the same time cannot be reproduced. This however also means that the PRR must reflect the situation when there are no collisions, as these are already handled by the model. When determining the PRR network-internal collusions must therefore be reduced to a minimum. Alternatively the effects of collisions must be eliminated from the PRR.

External Noise and CSMA If the channel is blocked by internal traffic, then this can and must be handled by the simulation, just like collisions. External sources however can be modeled in two different ways, either at the receiver, as part of the reception model, or as special node that generates noise and sends this noise via the propagation model (see section 2.6.4, page 29). The latter yields more realistic results as it allows reflecting correlations between nodes. Thus, if two nodes are placed close together they are more likely to receive or loose the same packets.

When modeling the noise sources as separate entities in simulation, then also packet loss caused by those is already handled by the model. Packets lost due to these noise sources must not influence the PRR. As, to my knowledge, no reliable method of extracting such information from a WSN exists, I will not speculate how the influence of these sources can be removed from the PRR.

Yet even when not explicitly modeling external sources, correlations in noise influence the PRR. This is the case when using CSMA protocols, which check whether the channel is clear (CCA) before sending. If there is no noise correlation between the sending and the receiving node such a check will not influence the PRR. However, if this is the case, then the packet loss can be reduced, because a packet is less likely to be sent while the receiving node is influenced by noise.

Consequently, when simulating a network which uses CSMA the PRR must also be acquired using CSMA. When doing so it has to be born in mind that the characteristic of the CSMA protocol used to gather the information must match that of the simulation. This applies to attributes like back-off times, but also to the algorithm that is used to decide whether the channel is in use.

Packet Size The larger a packet is, the longer the duration to send it and the more likely that the data gets corrupted. Most simple radio models have a constant PRR that is independent of the packet size. Others assume a fixed Bit Error Rate (BER) where each additional bit increases the likeliness of data corruption linearly. Or in other words: A packet twice as large is twice as likely to get lost. As shown by Lee [Lee06] as well as Chen and Terzis [CT10] in real world experiments the packet length has a minor impact on the PRR.

A possible reason can be found in [LCL07]. Especially in an office environment, but also in an active network, the main cause of packet loss are not random events, but by active interference by other devices. Opposed to effects handled as random bit-errors by a Gaussian BER model, the durations of interference and silence are relatively long. Thus, while the length of a packet does increase the chance of being interfered, it is of more importance whether the packet is sent in a period of silence or noise.

3.8.3.2 Receive Signal Strength Indicator (RSSI) / Signal Strength

Most radio chips and all supporting IEEE 802.15.4 provide a RSSI value. The RSSI however is only an indicator, therefore its significance depends on the radio chip used. As the popular CC-radio-chip series by Texas Instruments (TI), as well as some other chips, the RSSI can be mapped to the received signal power in dBm by adding a negative offset, these are often used as synonyms. This assumption must be treated with care, though. The data-sheets of those chips (e.g. [Tex04]) provide a typical offset, but state that is must be calibrated to the antenna used to get accurate data.

For many applications it is sufficient that the data is in dB, rather than some linear scale. Most effects seen in radio models are proportional and therefore do not need calibrated absolute values. For example many radio models assume that a signal is not interfered if the signal strength varies by a certain relative signal strength. For IEEE 802.15.4 this is typically 3 dB. This also means that it is not necessary that the collected data is calibrated, as the offsets in dB are constant.

The RSSI can be collected in two situations, when listening and when receiving a packet. Providing a RSSI of a packet is part of the IEEE 802.15.4 standard. For the CC2420 radio chip used for this work, this value is calculated for the first 4 bytes of data. Thus it must be assumed that the signal strength does not change too much during transmission. If this actually was the case, it is not possible to address this with the hardware at hand.

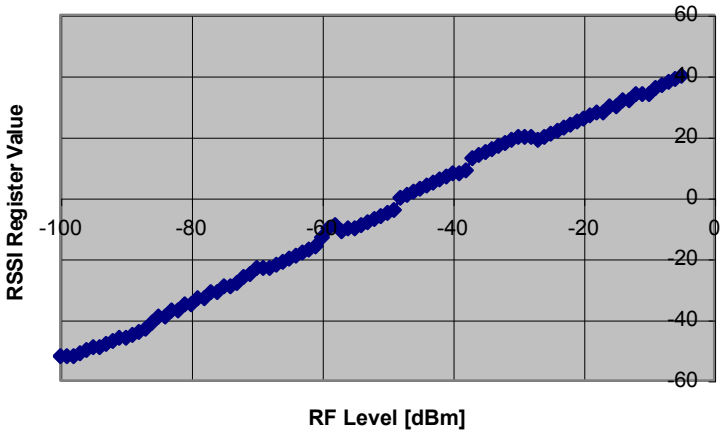


Figure 3.11: Typical RSSI value vs. input power for the CC2420 radio chip. [Tex04]

The background noise was already discussed in detail in section 3.7, page 63. As discussed there, the WSN domain lacks a model that can be parameterized using the resources provided by WSN nodes. Providing such a model can most likely improve the simulation results for noisy environments significantly.

Until such a model is developed, it is only possible to use an inaccurate averaged value, or a high resolution value that is saved using a side-channel such as the serial or an SD-card.

3.8.3.3 Link Quality Indication (LQI)

As the RSSI, the IEEE 802.15.4 standard only defines the LQI insofar that it is a value ranging from 0 to 255 where a higher value represents a better signal quality. The CC2420 datasheet [Tex04] vaguely defines it as "chip error rate". Therefore this value can only be collected using the actual hardware and then set as connection attribute that has no further influence on the simulation model itself. The simulated software however can of course make decisions based on this value, assuming that the other connection-attributes, also collected by the hardware, correlate to that value.

3.8.3.4 Delays / Time of Flight (TOF)

Some radio models support delays to reflect the time caused by the distribution of radio waves. While the delays of the TOF are measurable, and can even be used to measure the distance between nodes [Maz+11], their impact is rather marginal. The radio signal travels about 18 m/tick for a 16 MHz clock and almost 75 m/tick for a 4 MHz as used by the Sky mote. Under normal circumstances other effects like signal and packet processing on the radio chip, clock drift and interrupts have a much higher impact than the TOF [Pet+12].

3.8.3.5 Location

Unless the nodes "know" where they are, acquiring the location is of minor importance for the simulation results, as measured network attributes are of much higher accuracy then anything derived from the location. The location can be nice "eye-candy" when visualizing the network, though.

3.8.4 Conclusion

Although the data that can be collected from the deployment without extra hardware is limited in resolution and accuracy, there is a lack of alternatives. The overhead of using specialized hardware is huge and due to the temporal changes its additional accuracy is questionable. Using simulation can help in a pre-deployment stage, but is too inaccurate due to the complexity of radio propagation. And even then reference measurements are needed to calibrate the model.

Based upon these considerations I come to the conclusion that trace-based simulation is the only option to gain reasonable simulation-supported optimization results. In the next chapter I will therefore present RealSim, a toolchain to support the mapping of deployments to the simulator.

3.8.5 Outlook: Continuous Monitoring

While the presented solution currently only targets the deployment phase, it should also be applicable to a running system. However instead of generating the data, the data that is sent over the network tp provide its service can be exploited. There are however some challenges for this to work reliable.

Connection-Related Attributes While the RSSI and the LQI of the neighboring nodes can easily be extracted from the received packets, acquiring the PRR is not as trivial. There are two reasons for this. Fist, the sender knows how many packets were sent, while the receiver knows how many were successfully received. In the presented solution this was solved by adding an ID to each packet. Of course the network stack can be extended in a way that the header of the packets sent is extended by such an ID. Alternatively the number of sent packets are part of the statistics generated at the sender and the information is later merged once all data was collected.

When using RDC mechanisms to reduce the energy consumption and increase the life of the network, additional issues must be taken into consideration. When the radio of the potentially receiving node is turned off, the packet cannot be received. This effect must however not influence the PRR as it is already part of the model. To explain this issue in more detail, suppose all packets are received if the radio is turned on. If 99 packets are lost while the radio is turned off and a single packet is received while the radio is turned on, the PRR will be 1 %. The simulation model will however only test against the PRR when the radio is turned on. In the simulation again only 1 % of the packets sent are received while the radio is turned on. With a link-level PRR of 1 % this results in an node-to-node PRR of 0.01 %.

When using sender-initiated RDC protocols, each node turns on its radio within a certain time span. The sender repeats sending the data for such a time-span until it gets an acknowledgement from the receiver. Thus, the PRR must not be calculated based on the total number of packets sent by the radio, but based on the number of packets to be transmitted.

For receiver-initiated probing RDC protocols work the other way round. If a node wants to send data, it waits until it receives a beacon from the node it wants to send data to. It then has a certain time slot to send the data. It can therefore be assumed that the radio is turned on while sending the packet.

However, all probing protocols have in common that the likelihood of collisions is higher, because the sending nodes compete for a reduced amount of reception time. Also, it is only possible to reliably acquire data for connections that are actually used. Nodes that are not specifically addressed by a sending node, might not know of that node, because the radio of the receiving nodes is always turned off when the sending node is transmitting. This can only be mitigated by sending special broadcast packets that reach all neighboring nodes. Some protocols do this anyway to update their neighborhood table.

Most RDC protocols use techniques that are more complex than the basic concepts described. These specifics must also be considered when calculating the PRR. All in all, to allow an accurate survey of the network, it is necessary to adjust the monitoring technique to the network stack and the usage scheme of the applications running on the node.

Background Noise Opposed to the collection of characteristics of the network topology, the background noise is rather easy to survey. When using a sender-initiated RDC protocol the nodes turn on their radio once in a while to listen for potential packet. Similarly nodes using a receiver-initiated RDC protocol, have the radio on while they wait for the receiver to send its beacon. And even when using a TDMA scheme, turning or keeping the radio on for a little while longer can easily be tolerated without too much impact on the energy consumption. These times can be used to sample the background noise.

If a model is found that can be configured to more accurately model the background noise, it is likely that the radio must be kept on to acquire a continuous stream of samples. However, as of now no such model exists, and it can therefore only be speculated what kind of scheme is needed to make the noise measurements.

4

RealSim: Mapping Deployments to a Simulator

To evaluate the approach of optimizing Wireless Sensor Networks (WSNs) by mapping a real network to a simulator, two tool chains were developed: RealSim and DryRun. While the RealSim tool-chain supports mapping the actual network to the simulator, DryRun explores a parameter space and will be presented in chapter 5, page 91.

To map the network to the simulator, the network must first be surveyed. The collected data can then be replayed using a WSN simulator – in this case Cooja [Öst+07]. The tool chain is split into the following components:

1. **Survey Application:** To survey the network, a special application, which runs on the nodes, sends out beacons containing the ID of the sending node and an ascending packet ID. Based on this information the receiving node can generate statistics for all neighboring nodes: The Packet Reception Ratio (PRR) is derived from the packet ID. As the ID is ascending, lost packets can be detected, if an ID is skipped. Receive Signal Strength Indicator (RSSI) and Link Quality Indication (LQI) are provided by the radio chip.

   Additionally, the RSSI is sampled to monitor the background noise. As discussed in section 3.8.2, page 74, acquiring more than these three attributes is too complex for the use in WSNs. Besides, the selected radio model does not support more metrics without significant extensions. The collected data is aggregated over a constant period (epoch) and can be either printed directly to the serial interface for testbed usage, or sent to a sink node, which will then print the received data to the serial.

   As basis for the application Contiki OS was used. The complexity of the program logic was kept simple enough to be easily ported to any other WSN Operating System (OS).

2. **Data Preprocessing:** After collecting the survey data, it is preprocessed. This is necessary because the data can be provided in different formats, depending on the technique used to collect the information. During the preprocessing step, the relevant information is extracted from the collected outputs and plausibility checks are run.

   The nodes also do not have synchronized clocks. Printing the information directly to the serial, using the time stamp provided by the host system is accurate enough. When collecting the data through a sink node, the data is only annotated with the time stamp of the sending node. This must be put in correlation with the global time.

   The resulting data is saved in a format that is optimized to be processed by the RealSim plugin for Cooja. Alternatively it is possible to save the data in a more generic format that allows further analysis using other tools such as R.

3. **Cooja Plugin:** The RealSim plugin for Cooja allows replaying the previously collected data in the simulator. The plugin reads the trace from the preprocessed file and adjusts the Directed Graph

Radio Medium (DGRM) accordingly. As this is done in simulation time, there are no real-time constraints.

In the following I will present the three tools, their challenges, requirements and design decisions.

4.1 Survey Application

As introduced, the survey application is responsible for acquiring information about the network. It collects a time series of the PRR, RSSI and LQI of each neighbor. The result is a directed graph of the network topology. The application also acquires the background noise at each node.

The survey applications is assembled of three components:

- Sending packets
- Processing of received packets
- Publishing data via serial or sink node

Before explaining the exact workings of these components I will give an overview of the methodology used to survey the network.

4.1.1 Methodology

The measurements are divided in epochs of constant duration. Their duration is the same for all nodes of the network, yet are relative to the node's startup time and therefore not synchronized. During each epoch a fixed number of packets containing a sequence number is sent. The packets are sent using CSMA (Carrier Sense Multiple Access) as discussed in chapter 3 (Analysis, page 39). In this work I used epochs of 80 s, sending 8 packets per epoch.

The RSSI and LQI of each packet are summed up and divided by the number of packets received during that epoch to calculate the average. The PRR is derived from the number of received packets and the number of packets that should have been received within an epoch.

Due to the fixed number of packets per epoch it is possible to derive a packet's epoch from its sequence number. This has multiple advantages for the processing at the receiver side. Once a packet with a sequence number from the next epoch is received, no more packets from the previous epoch can arrive. The statistical data for that epoch can be calculated and intermediate data discarded. If the last packet of a node arrived more than an epoch ago, no more data from that epoch can arrive, allowing calculating the statistics and discarding intermediate data.

To avoid effects caused by regular intervals, the packets are randomly distributed over the duration of an epoch. For each epoch this is done a-priori, to have a statistically equal distribution. A minimal pause between packets is kept to allow the CSMA algorithm to send the packet. If the randomly generated point in time violates this rule, a new one is chosen instead.

The RSSI values to determine the background noise are sampled in random intervals ((350 ± 50) ms). As the number of samples is quite high and solely the average is calculated, the number of samples can differ from epoch to epoch. The radio chip indicates whether it is currently receiving a packet. If this is the case, the sample is discarded. As the number of samples is large enough, the impact of the RSSI value being sampled while receiving a packet on the average will be insignificant, yet it does no harm filtering these out.

Discussion: Resolution of the Background Noise Figure 4.1 shows the statistical data of two selected nodes with different background noise characteristics. The data was collected as part of the analysis presented in section 3.7, page 63. It is aggregated over 1 s respectively 8192 samples. One can see that the mean is stable and even the median has only a very small jitter. This shows that the changes of the background noise either have a much higher frequency than 1 Hz or a lower frequency than the used 80 s epoch. Thus the only way to improve the quality of the data is to increase the resolution to a value higher than 1 Hz. As discussed in section 3.7, page 63, this is not practical as it produces more data than can be processed by a typical WSN. The only alternative, a model that can describe the noise with the resources available on the node, is, to my knowledge, currently not available.

4.1.2 Technical Implementation

In the following I will describe the technical implementation in more detail. As already noted I built the application using Contiki OS as basis. For the experiments version `fe0a0423` [Con] was used, however

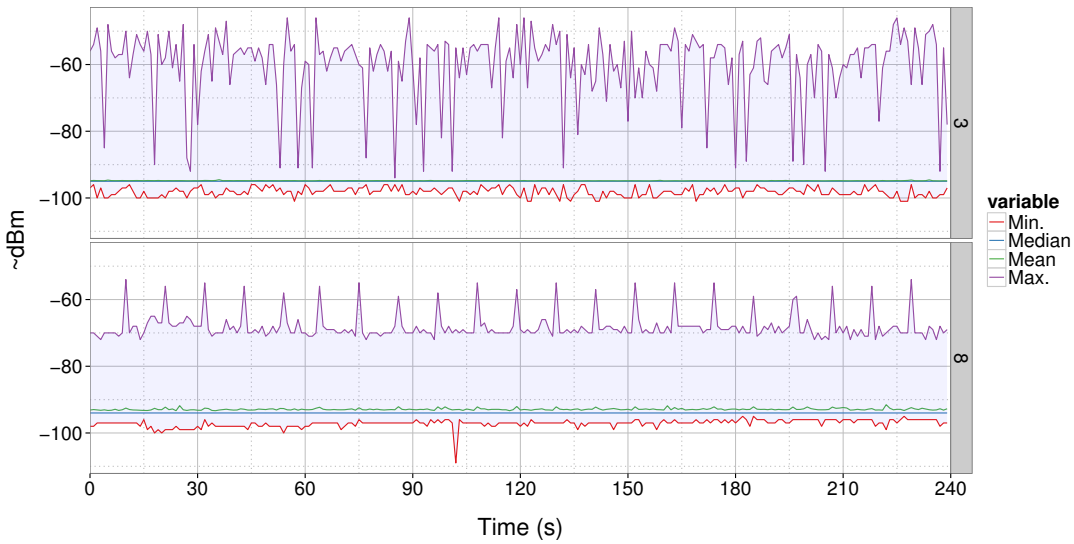


Figure 4.1: The graph shows the statistical data of ③ and ⑧, which have different background noise characteristics. The statistical data is aggregated over 1 s respectively 8192 samples.

migrating the software to newer versions should be possible with minimal effort. The default configuration for TMote Sky nodes is used. The only change made, was that the network stack was adjusted to use a simple CSMA Medium Access Control (MAC) protocol. Opposed to ContikiMAC, no Radio Duty Cycling (RDC) is involved and the radio chip was therefore always listening and being able to receive any packet possible. In the default configuration Contiki's implementation of the CSMA protocol makes three attempts to send the packet when the noise level is beneath the Clear Channel Assessment (CCA) threshold. Unfortunately no statistics on how often all three attempts fail were collected during the experiments. Other changes to the OS were required to suit the requirements of the application and will be referenced accordingly.

4.1.2.1 Initialization

The initialization is mainly architecture specific, such as preparing data structures that are needed at a later point in time, initializing ring buffers, registering events and timers at the OS and initializing the network stack to send and receive broadcast packets. The CCA value is set to a compile-time configurable value. Support to change the CCA during experiments is currently not implemented.

4.1.2.2 Sending Packets

Sending of packets is split up in two steps:

1. Create schedule for the next epoch.
2. Send beacons at the scheduled intervals.

The scheduling algorithm works as follows:

1. Generate an epoch-based offset using the Pseudo Random Number Generator (PRNG) that is within the time frame of the epoch.
2. Check if the minimum time between two packets is kept. This includes the last packet of the previous epoch; otherwise discard that offset.
3. If the required number of offsets is generated, break.

Afterwards the deltas of the offsets are calculated to simplify setting up the timer.

A timer provided by the OS is used to trigger sending of packets. This however is no real-time timer that runs in an interrupt context, but a signaling timer that is checked when the cooperative scheduler runs. This is acceptable as this adds additional randomness to the exact execution timing. Care was taken that the timing does not depend on scheduling decisions and execution timing, but is always relative to

```
struct neighbor {
    struct neighbor *next;
    linkaddr_t addr;
    unsigned long last_seen;
    uint16_t last_seqno;
    uint8_t recv_count;
    int16_t rssi;
    uint16_t lqi;
};
```

Listing 4.1: Neighbor struct to save information about neigboring nodes.

the previous trigger[1]. As payload the packet contains the ID of the sending node and a 16 bit sequence number. The integer overflow of the sequence number is intended and handled at the receiver.

Discussion: Randomness To avoid the effects of deterministic behavior the beacons are spread over the epoch using the PRNG provided by the compiler or rather its C-Library. In case of GCC it is provided by the GNU LibC standard C library. Already in 1988 it was considered to be a rather bad PRNG, especially for the lower bits. [PM88]

As the use of PRNG rather targets avoiding a periodic schedule, than providing statistical randomness, even suboptimal PRNG should be sufficient to schedule sending packets. The same applies to the PRNG always being initiated with the same seed at startup, causing each node to have the same schedule for each experiment. Inter-node effects caused by the schedule not changing between experiments are dampened by starting the nodes in a random order and clock-drift. Also, effects caused by interrupts, scheduling and the CSMA algorithm add slight variations.

4.1.2.3 Receiving Packets

When a packet is received by Contiki a call-back handler is called. To keep this handler as short as possible, it only extracts sender ID, sequence number, RSSI and LQI from a received packet and writes it into a buffer. It also schedules a task that processes the data from the buffer.

The handling function generates the statistical data. For each neighbor a data structure is allocated (listing 4.1). A linked list is used and the correct data set is identified by matching the address (`addr`). This data-structure is used to save information about when the last packet was seen (`last_seen`) and its sequence number (`last_seqno`), the number of packets received (`recv_count`) in the current epoch of the sending node, and the sum of the LQI (`lqi`) and RSSI (`rssi`) values in this epoch.

Once a packet with a sequence number of the next epoch is received, statistics for the last epoch are calculated: From the received data the PRR and average RSSI and LQI are calculated. Statistics are also generated when the last packet arrived more than two epochs ago. The time frame of two epochs was arbitrarily chosen to safely tolerate possible jitter.

RSSI The RSSI provided by the CC2420 radio chip is calculated over the first 8 symbols of the packet. Its value maps to the signal strength in dBm with an offset of ≈ -45. The exact value however must be found using measurements, yet, as already discussed in 3.8.3.2, is of minor relevance. For the application at hand not having accurate absolute values is however not a problem, as the data is not used to make calibrated measurements, but to configure the simulator. The simulator will add 45 to the configured value and place the result in the appropriate register of the simulated radio chip. As all other decisions, for example the CCA, are made based on this value rather than the actual signal strength, the simulated radio chip will behave just the same as the real radio chip.

4.1.2.4 Background Noise

The task to acquire the background noise is rather simple. In intervals that have a random component (e.g. (300 ± 50) ms) the RSSI is sampled from the radio, unless it is currently receiving or sending a packet. For each epoch the maximum, minimum and average value is calculated. Calculating the median is not possible using the available resources (see section 3.5.1.1, page 50).

[1] To achieve this adjustments to the OS had to be made (`1f2c226d` [Con]).

4.1.2.5 Data Output

The calculated statistics of the neighbors or background noise are passed to a handler that either prints them to the serial or sends them to the sink. When sending the statistics to the sink, the data is annotated with a time stamp and buffered until a packet is filled to minimize the traffic on the network. To reliably associate the data with the correct node every output also contains a node's ID.

As none of the standard protocols provided by Contiki support reliable transmission to the sink, this was implemented at the application layer. Therefore the assembled packet is kept in a buffer until the sink acknowledges the reception. For the experiments the contiki-mesh protocol[2] was used, but any other protocol supporting multi-hop transmissions should be just as suitable. Each packet sent to the sink also has a header containing the sending node's ID, the current time of the sending node and a sequence number. The time is added to better correlate the statistics collected by the different nodes while processing the data. Once a packet arrives at the sink, it prints the data to the serial and acknowledges the packet with that sequence number to allow freeing the memory at the sending node. If no acknowledge arrives within a certain time frame, in the experiments 15 s were used, the packet is resent.

On the host side the data received from the serial is annotated with a time stamp and saved into file for later processing.

4.1.3 Processing the Captured Data

As already noted, the output can happen in two ways: Directly to the serial, or via a sink node. If the data is printed directly to the serial, the time stamp added by the host is used to synchronize the different nodes. If the data was sent to the sink, the local time of the nodes is put in relation to the time stamp added by the host. The smallest delta, which is provided by the fastest packets, is used to put the node's clock in relation to the global time. Due to the long duration of epochs, deviations of a few seconds are considered tolerable.

The processed data is saved in a format where the time stamps are relative to the first received packet. The original time stamps of the captured data are lost. This file can be directly read by the RealSim plugin for Cooja.

The RealSim plugin for the Cooja WSN simulator always processes the whole configuration file, creating the necessary structure in memory. This has the advantage that the data in the file does not have to be in chronological order. Loading a several-hour trace to run a 20 min experiment adds an unnecessary overhead. The plugin also does not support starting the trace with an offset. For this use case an application called Rs2Rs is provided, which allows selecting a subsection of the converted trace by providing offset and duration. During the conversion care is taken that the configuration at the new time 0 corresponds to the state prior to the starting time of the cropped trace.

Processing of live data is implemented in a less sophisticated way. If new information about a connection is received, this is directly forwarded to the RealSim plugin. In the live-mode a connection is removed, if it has not been updated for a configurable amount of time (e.g. 4 min).

4.1.4 Related Work

Trident [Chi+11; Ist+14] is a tool to run connectivity experiments. It however was too sophisticated for the early experiments and did not support monitoring the background noise and live monitoring. Due to the simplicity of the survey app, extending Trident to support the necessary features would not have been reasonable.

4.2 Replaying Data

Replaying data in the simulator is the second major component of RealSim. To achieve this, the Cooja WSN simulator was extended with a plugin that allows feeding the previously traced data into the DGRM.

In the following I will first give a brief technical introduction into the Cooja network simulator insofar, as it is required to understand how the plugin integrates.

[2] To allow an efficient usage, the protocol was enhanced with the patches `b1dbc204` [Con], `a03e899c` [Con]

4.2.1 The Cooja Network Simulator

The Cooja WSN simulator is basically a discrete-event meta-simulator. It allows plugging Network Level Simulators (NLSs), System Level Simulators (SLSs) and Instruction Level Simulators (ILSs) together. It provides multiple network models that can be used to let the nodes interact. Examples are the simple Unit Disk Graph Model (UDGM), the DGRM used in this work, as well as a multi-path ray-tracing radio medium.

The radio medium can be accessed from any of the abstraction layers provided by Contiki at the same time. It therefore possible to have a Cooja-Node (NLS), a native Contiki-Node (SLS) and a Sky-Node (ILS) interact with each other. The SLS is currently only supported by Contiki. To provide ILS Cooja falls back to Avrora [Avr] and MSPSim [MSP], which both provide multiple WSN node models. As all are written in Java, the integration works seamlessly using an adoption-layer that maps the execution steps of the other two simulators to Cooja's event-queue. Currently 13 different platforms, including the popular MicaZ, Z1, Sky and ESB nodes, can be accessed using Cooja.

During the development care was taken to implement an object-orientated, extendable design. Many features are implements using a plugin interface. This also allows to easily add new radio mediums as well as node types. The plugins are loaded at runtime and do not need to be placed in a specific folder. Instead it is possible to configure the simulator to load the plugins from specific folders, which allows out-of tree development, not having to adjust the original code.

Mote Types In Cooja nodes are called motes. Cooja also distinguishes between a mote type and a mote. A mote type is the combination of a certain mote (e.g. TMote Sky) and software. This makes it easy to create multiple nodes running the same firmware. During the initialization Cooja does adjust the ID chip or the flash-memory to reflect a pre-configured ID. This is important, as the nodes could otherwise not address each other.

4.2.1.1 Required Adjustments

Due to the RealSim plugin utilizing the simulator in unprecedented ways the following adjustments were required. Both are now part of the Cooja code-base.

- Normally nodes are added to the simulation either when loading a configuration or using the Graphical User Interface (GUI). When adding a node while the simulation was running, it was only accessible after a scheduled event, which created the node, was executed. The plugin however needed to access the node immediately, to configure it to the given requirements. This was fixed in `a3eb4238` [Con].

- Starting a simulation required at least one node to exist. Yet with the RealSim plugin adding a node at a later point in time became possible. As the simulator checks whether there is at least one event in the event queue anyway, the check was removed in `7bca2e23` [Con].

4.2.2 The Directed Graph Radio Medium

The DGRM allows setting the RSSI, LQI, PRR and delay for each connection. For the radio model only the PRR and delay are relevant. RSSI and LQI are solely passed to the radio driver, which however evaluates the RSSI. The same applies to the background noise, which can be set per node.

The reception logic of the DGRM works as follows. For each node that has a connection to the sending radio a number of checks are done. If any of the checks fail the packet is not received.

1. Check the radio is turned on.
2. Check whether the radio is not already marked as interfered.
3. Check whether the radios use the same channel.
4. Check whether that radio is already receiving data. If so, the radio is marked as interfered. The radio driver is responsible to cancel the ongoing reception.
5. Check the PRR against a number from the PRNG.

Note that the capture effect is not considered. This is a shortcoming of the implemented model.

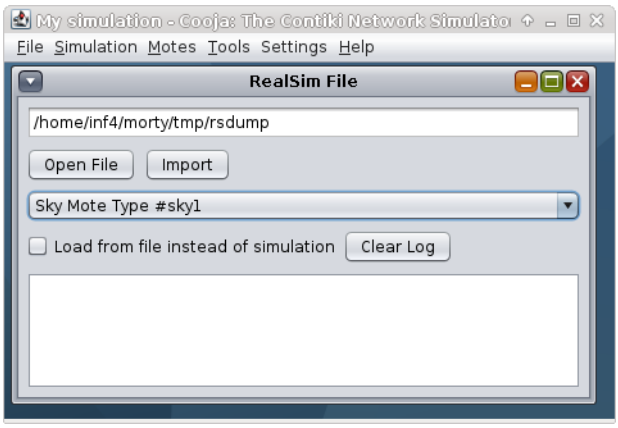


Figure 4.2: The GUI for the RealSim File interface. It allows setting an input file, the default node type and configuring whether its configuration should be loaded from an external file or saved as part of the simulation when opening a simulation configuration. At the bottom it provides a textbox with log-output.

4.2.2.1 Required Adjustments

While working with the radio medium the following changes were made to successfully run the experiments. All fixes became part of Cooja.

- Correctly set the signal strength at the receiving radio. Before, the values configured in the DGRM were ignored. (`9b1fb12a` [Con])
- Add support to set the LQI. This was not possible at all. (`80e7a6fd`, `bed38779` [Con])
- Add support for background noise. Before all nodes experienced perfect silence when no node was sending. (`c22f1ea6`, `86320d74`, `1ef80859`, `123a7081`, `4deb6872` [Con])[3]
- Limit the RSSI ranges used to calculate the color used to visualize the connection properties. Before, if the values were too high, an unhandled exception was thrown, causing the simulation to crash. (`3c3b25a5` [Con])

4.2.3 The MSPSim Simulator

Cooja integrates TMote Sky nodes, which are used as reference platform in this theses, utilizing MSPSim [MSP] as ILS. MSPSim was specifically developed to be integrated into Cooja and therefore supports different WSN nodes, yet does not allow network simulation natively. It is nonetheless possible to simulate a single node without the need of a network simulator.

To support those platforms, MSPSim must not only emulate the micro controller, but also all peripherals that are to be reflected in the simulation. These include digital sensors for light and temperature, external memory and multiple radio chips.

During the experiments I was able to trace the cause of simulation and testbed behaving differently back to a bug in MSPSim: Although it was possible to set the CCA threshold, which decides whether the incoming signal is strong enough to consider the channel blocked, this register was ignored, causing simulations with different CCA thresholds to yield the same results. This was fixed in `0a13f999` [MSP].

4.2.4 The RealSim Plugin

The RealSim plugin is split into three components. An abstraction layer for Cooja and two interfaces: One to read the data from a file and one to handle live data. The abstraction layer was added, to reduce the shared code between the two interfaces and make their code easier to maintain.

The plugin only supports adding and removing nodes, as well as adjusting the DGRM configuration. The rest of the simulation environment must be set up manually. Specifically at least one mote-type must be created to be used by the plugin when creating new nodes.

4.2.4.1 The File Interface

The file interface is used to run experiments based on traces. Yet it can also be used to manually set up certain scenarios with changing connection properties. Figure 4.2 shows the GUI used to set up an experiment. As input it takes a configuration file, which will be introduced in the following section, containing a setup and actions with timings when these are to be executed. An event-handler is registered at the simulator for the first event. This handler takes care of adjusting the simulation according to the configuration and reschedules itself for the following events. Using the drop-down menu the default node can be selected. The mote-type can also be configured using the configuration file.

All settings are saved as part of the simulation configuration. It is possible to also save the loaded events, or to just save a reference to the file containing the trace. The latter allows adjusting the trace without the need to modify the simulation configuration.

4.2.4.2 The RealSim File Format

All events to be executed in the simulation have the following format:

<Time in ms>;<Command>;<Parameter1>;<Parameter2>;<...>

Values are separated by semi-colons. The first value is the time at which the command is to be executed in ms. While the simulation internally uses µs, ms seemed good middle ground between keeping the values within a reasonable range without the need to support units (e.g. "100ms"). The time is followed by a command and its command-specific parameters. As the file is completely parsed while loading, the order of the events can be arbitrary.

The following commands are supported:

- `addnode;<nodeid>[;<node type>]`
  `addnode` adds a node with the ID `nodeid`. It is possible to optionally set a `node type`, as discussed below. If a node with the same id already exists, the command is ignored.

- `rmnode;<nodeid>`
  `rmnode` removes the node with the ID `nodeid`.

- `setedge;<srcID>;<dstID>;<PRR>;<signal>;<LQI>`
  `setedge` configures the edge that points from `srcID` to `dstID`. If one of the nodes is missing, this command is ignored. PRR is set in %. The `signal` strength is set in dBm, which corresponds to the RSSI value −45 for the CC2420 radio. As the LQI is an arbitrary value is passed on without any adjustments.

- `rmedge;<srcID>;<dstID>`
  `rmedge` removes the edge from `srcID` to `dstID`. This is not the same as setting the PRR to 0: Even if the packet cannot be received, the signal still influences the radio model, typically by causing interference.

The file can also contain the following variations of the `nodetype` command. While the events can be in arbitrary order, the `nodetype` applies to `addnode` commands that follow.

- `nodetype`
  Without a parameter the default node type is reset to the default type selected in the drop-down of the RealSim GUI.

- `nodetype;<Desciption of nodetype>`
  For each node type Cooja allows setting a description, which is rather used as a name. If multiple node types were created in the simulation, all following `addnode` commands will create nodes of that type by default.

- `nodetype;<Desciption of nodetype>;<id>;<id2>;...`
  It is also possible to set the node type for specific node ids. This overrides the other configuration options, yet only applies to nodes created after this line.

[3]These changes were not made by myself, but under my supervision.

The logic of interpreting the RealSim configuration files allows for multiple ways of setting up a configuration.

- **Manually setting up the nodes**
  As the nodes are only crated by the `addnode` command, if they do not yet exists, they can be manually created within the simulation. The `nodetype` parameters therefore do not influence the simulation, unless the node is first deleted using `rmnode`.
- **Using a "global" configuration**
  By explicitly setting the node type for each node at the beginning of the file using the `nodetype` command, the order of all following commands becomes irrelevant. The following example demonstrates this:

```
nodetype;sinkNode;1
nodetype;defaultNode;2;3;4
0;addnode;1
40;addnode;2
100;setedge;1;3;90;-83;96
200;addnode;3
500;addnode;4
```

  Alternatively it's also possible to achieve the same result by only specifying a specific node. This has the advantage of not having to know how many nodes will be created. In the following example node 1 is created using the `sinkNode` type while nodes 2 to 4 are created of the default type selected using the GUI.

```
nodetype;sinkNode;1
0;addnode;1
40;addnode;2
100;setedge;1;3;90;-83;96
200;addnode;3
500;addnode;4
```

- **Concatenating node-specific configurations**
  It is also possible to concatenate node-specific configurations as indicated by the following example. This allows to easily remove a specific node.

```
nodetype;sinkNode
0;addnode;1
100;setedge;1;3;90;-83;96

nodetype;defaultNode
40;addnode;2
1000;setedge;2;3;80;-88;75

nodetype;defaultNode
200;addnode;3

nodetype;defaultNode
5000;addnode;4
```

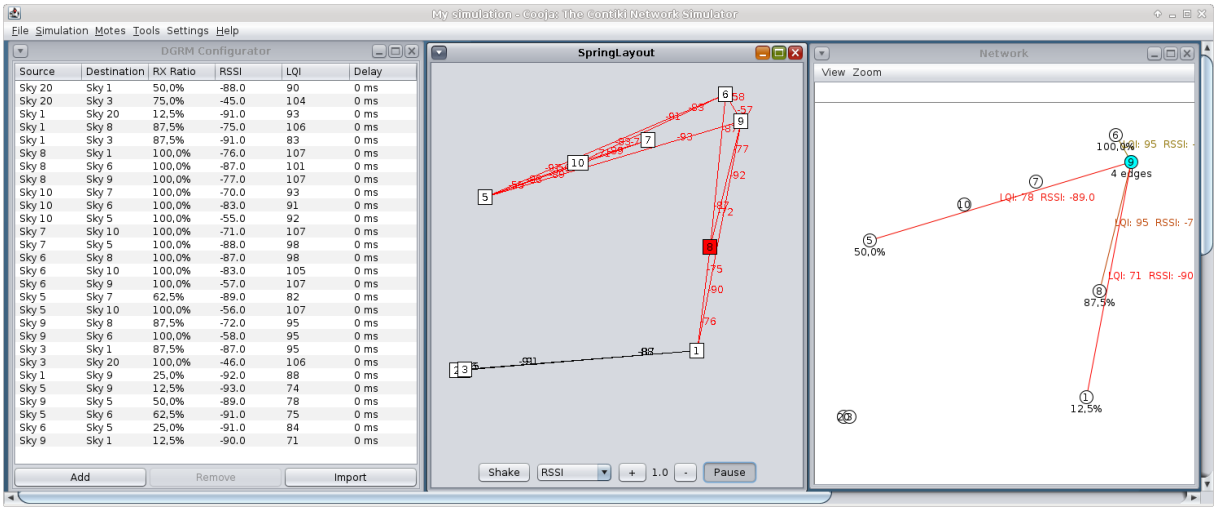
4.2.4.3 The Live Interface

The Live interface (fig. 4.3) allows adjusting the simulation based on live data. Using the GUI it is possible to configure a Transmission Control Protocol (TCP) port for the client to connect to and select a default node type. Opposed to the File interface, nodes are not explicitly created, but created automatically when they are referenced.

The protocol is line-based and held very simple:

```
<timeout> <srcID> <dstID> <PRR> <signal> <LQI>
```

Figure 4.3: The GUI for the RealSim Live interface. It allows setting a port to listen for incoming connections from the client as well as a default node.



-

Figure 4.4: Placing nodes using the SpringLayout plugin in Cooja. The left window shows the DGRM Configurator, which allows configuring the edges of the graph. The middle window shows the SpringLayout plugin. The red background indicates that node 8 is fixed. Its position is therefore not altered by the spring layout algorithm. The colors give an indication whether the spring is under pressure or being pulled apart. The Shake button repositions the nodes randomly to escape an unsuitable minimum. The drop-box to the right allows to select the algorithm to determine the spring length. The length factor can be set with the "+" and "-" buttons. As the algorithm is iterative and has no termination criterion, the positions of the nodes are likely to oscillate or drift. It can therefore be paused using the "Pause" button. The right window shows the normal network visualizer, which allows different visualizations of the network. Currently it shows the connections of node 9.

It is basically the `setedge` command of the File interface with a prefixed `timeout` in s. If the edge is not updated within that timeout, the connection is considered lost and removed. Support for the background noise was not added yet, but extending the protocol is possible.

A simple client connects the gateway node's serial to the plugin's network port. It parses the data from serial interface and forwards the updated connection properties to the plugin.

This mode is suitable when deploying a network or wanting to monitor live changes. Opposed to the File interface, simulation time is not correlated to the real time and any change is directly forwarded to the simulator. Thus, the data is updated as soon as it arrives at the sink. It can therefore also be used to just visualize the live data, without simulating nodes.

4.2.5 The SpringLayout Plugin

When using the DGRM, only the properties of the connections between the nodes are of relevance. The nodes also have a position, as this is required by Cooja, but it has no impact on simulation results. RealSim therefore places the nodes randomly.

Yet, especially when learning about a network, it is nice to place the nodes so that they can resemble an actual network, placing nodes with good connection close together and those with bad connections further apart. To accomplish this, a plugin was developed for Cooja to position the nodes using spring-based

positioning. A screen shot of the plugin is shown in fig. 4.4. The plugin uses an iterative algorithm that is based on the idea that each edge is replaced by a spring. The length of the spring is defined by the average connection properties and can be selected in the plugin (PRR, RSSI and LQI). The stronger a spring is stretched the stronger two nodes are pulled together, or pushed apart if the spring is under pressure. Based on the forces that push onto a node the algorithm iteratively moves the node around until an equilibrium is found. Due to the discrete nature of the algorithm, it is likely that the whole system will swing. To compensate for this, the movement of the nodes is damped if they change the direction in comparison with the previous stop.

a) Without repelling force: No force pushes the center node away.

b) With repelling force

Figure 4.5: Spring layout with and without additional repelling force

To improve the positioning, an additional force component was added so that all nodes repel each other. The motivation is visualized in fig. 4.5. When only using the springs, nodes that are only connected to one node can freely float around that node without being influenced by other nodes. This can cause them to be positioned close to other nodes, which they do not actually have a connection to (fig. 4.5a). If the nodes are repelling each other, leaf nodes are pushed to the edge of the graph (fig. 4.5b).

The screen shot shown in fig. 4.4 shows one of the limitations of this approach. Node 7 is only connected to node 5 and node 10, yet is positioned close to nodes 6 and 9. This is caused by node 10 having a significantly better connection to nodes 6, 7 and 9 than to node 5. The repelling forces of nodes 7 and 9 are not strong enough to compensate for that.

Due to the simplicity of the algorithm, the node cannot escape a local minimum. Nodes can therefore be dragged with the mouse to position them more suitable. It is also possible to fix a node at a position using a double-click. Currently it is not possible to adjust the algorithm to calculate the length of the springs without adjusting the source code. It is however possible to set a factor.

4.3 Verification

When verifying RealSim it must be distinguished between verifying the components developed as part of this work and verifying the whole system which includes the Cooja simulator and Contiki OS. The different RealSim components were well tested during the development and the evaluation:

The RealSim client was tested by printing debug information of the raw data collected to the serial and verifying that this corresponds to the aggregated data. The processing of the data was checked by selecting random samples and verifying that the processed data corresponds to the input data. The RealSim plugin was tested by running the RealSim Client in the simulator and checking that it yielded the same values, or in case of the PRR similar values, as set as link properties.

For RealSim to work however, the other components have to work correctly, too. During this work multiple bugs were revealed by comparing results yielded from the simulation and the testbed. These include hardware-issues (e.g. section 3.7.3, page 72) and limitations and bugs in the simulator (e.g. section 4.2.2.1, page 85) and the emulator (e.g. section 4.2.2.1, page 85).

4.4 Discussion: Limitations and Outlook

The conceptional limitations were already discussed in depth in the suggested approach (see section 3.8, page 73). Using RealSim it is possible to compare results from the testbed and the simulation. This allowed uncovering shortcomings in the simulation software. That they became visible, shows that bugs and insufficient models had a much higher impact on the outcome of a simulation than the deficits of the approach.

There are however some open points that are not solved, yet could significantly improve the results and usability of RealSim:

Noise Model The matter of the background noise, which has been looked into in section 3.7, page 63, is an unsolved problem. Currently there is no real model to allow modeling the background noise, and the current approaches rely on tracing the noise at a high rate and analyzing it using a powerful computer. Due to the limited resources of WSN nodes neither collecting the necessary amount of data, nor processing it, is a suitable approach for typical deployments.

Clock Drift With the latest versions of Cooja clock drift became supported (see section 3.6.5, page 58). Using concepts used for clock synchronization it should be possible to determine the relative clock. This was however not available while running the experiments and needs further investigations, especially as the RealSim client needs to be extended to also monitor the relative timing between nodes.

Continious Monitoring Being able to monitor a network while running the normal application allows using RealSim not only during the deployment stage, but also while the network is in production. How this can be done is discussed in section 3.8.5, page 77.

There are also two issues with the Cooja and MSPSim simulator that have been uncovered, but not yet been fixed.

Capture Effect The capture effect (see section 2.6.7.3, page 35) allows a strong signal to be received in the presence of a weak one. Supporting the capture effect does not only require adjustments to the radio model, but also to the emulated radio chip. Cooja currently lacks interfaces to propagate the information to the radio chip. Additionally, MSPSim's CC2420 model, but probably also other models, is currently not able to cope with this either.

Radio Chip Model / CCA MSPSim's CCA implementation of the CC2420 chip only compares the signal strength to the CCA threshold to decide whether the channel is clear. The default provided by the radio chip, however, also considers the channel in use when it is able to detect "valid" data (see also section 3.6.9, page 61).

In the evaluation the testbed often outperforms the simulation, especially in high traffic scenarios. The incorrectly implemented CCA and capture effect, are a likely cause for this. This has however not been verified.

5 DryRun: Testing Configurations

Being able to run realistic simulations is only the first step. The high number of configuration options and their possible settings create a huge parameter space (section 3.2, page 42) that is practically impossible to explore manually. Even simple parameter spaces with only 2 configurations parameters can easily have thousands of data points. This cannot be handled manually. Thus it is necessary to provide tool support that creates experiments, executes them and collects their results.

Due to this need of tool support the DryRun tool chain was created. The central tool is the test generator. It allows setting up the different steps of an experiment and the possible parameters of each step. The different parameters are used to span up to a parameter space. For each entry in the parameter space a script is created that executes the experiment. Currently the test generator only supports this brute-force approach of simulating each and every possible combination.

The second important tool is the data collector. It extracts the relevant data from the different files that are generated during an experiment. Examples are sizes of memory sections, information generated by the simulator or by the software running on the nodes, which is for example printed to the serial interface. The data of an experiment is merged with the input configuration and then saved for further processing.

Finally, two tools were developed to support running experiments. The first is CoojaTrace. It is a plugin for Cooja that gives easy access to many parameters of the simulation, the nodes and the software running on the nodes. These can then be logged based on state changes, time intervals or arbitrary events.

The second tool is the Wisebed experiment tool. Wisebed is the software used to manage the different testbeds used in this work. It provides a web interface to run experiments manually and an Application Programming Interface (API) to control the framework using an application. While reference implementations were provided for the latter, these lacked the error handling that is required to automatically run hundreds of experiments without the need of intervention. The Wisebed experiment tool can be scripted and provides functions that handle most of the possible errors autonomously and support running simple checks and the collection of results.

5.1 Test Generator

The test generator supports the user in creating a test campaign. [Dry] The user defines the steps that must be undertaken to run the experiment. Such steps are for example linking or copying of files, building the binary image or running the simulator. The provided functions allow to easily configure different configuration options. After a campaign is configured, a shell script is generated for every experiment. The shell scripts then take care of setting up the necessary environment, executing the experiment and collecting the results.

Each campaign is organized in three folders that are associated with a certain phase of the campaign:

- **Setup folder** The setup folder contains the script used to generate the campaign and all files necessary to run it. These are the input for the test generator, simulations, the files to be compiled, etc.

```
implicit val exp = new Experiment

val lnk = new Link("/some/path/contiki", "contiki")
exp.addstep(lnk)

val files = new GetFile("src/*")
exp.addstep( files )

val mk = new Make
mk.addConf("CCA_THRESH",−54, −20, 2)
mk.addConf("TEST_RATE", 5, 30, 5)
exp.addstep(mk)

val cooja = new Cooja("test1.csc")
cooja.addRandRange(1, 50)
exp.addstep(cooja)

exp.generate
```

Listing 5.1: Example experiment setup

- **Campaign folder** The campaign folder is created by the test generator. It contains all the files needed to run a campaign. Each experiment's files are saved in a separate sub folder of the campaign folder. This includes the shell script, as well as some meta-information used to describe the experiment. After an experiment is run, the results are copied to the experiment's sub folder.

- **Execution folder** The script generated for each experiment creates a separate execution folder. It is used to set up a clean experiment environment for each experiment. After executing the experiment all relevant files are recovered from the execution folder and copied to the experiment's sub folder in the campaign folder. After executing an experiment, the execution folder can safely be deleted, or kept for analyzing the experiment using the generated intermediate files.

The motivation for this design is the following. If nothing else changes, the setup folder is sufficient to recalculate all results, as the same input generates the same experiments and therefore the same simulation results. Setting up the execution environments in a separate execution folder allows sharing the campaign folder using a network share while executing the experiment on the local machine, which provides better performance. Using a separate execution folder also eases cleaning up after a campaign is run.

The test generator is written in Scala [OA06], a programing language that combines elements object-oriented and functional programming. Scala code is translated to Java byte code and is executed by a JVM (Java Virtual Machine). One of Scala's features is that it comes with a built-in run-time compiler. It is therefore possible to compile and run code within a program, and even access the currently running program's state. The test generator is therefore assembled of a library to generate the experiments and the run-time compiler. When executed, the test generator adds some syntactic sugar to the file passed and executes it. This approach gives the test script the power of a programming language and saves the trouble of having to write a parser.

Listing 5.1 shows an example script that based on the scripts used for the evaluation. In line 1 a new `Experiment` is created using the default options. The `implicit` keyword is required due to certain features that were used to simplify writing an experiment. The `val` keyword creates a new constant. The `Experiment` class is the central class than "manages" a test campaign. All steps of an experiment are registered at the `Experiment` object using the `addstep` function and will be executed in that order during the experiment.

The `Link` class (line 3) generates the necessary commands to create a symbolic link from `contiki` in the experiment environment to `/some/path/contiki`. In the next line (line 4) it is added to the experiment as first step.

The `GetFile` class will copy all files from the `src` folder to the build environment (line 6). This folder can for example contain the experiment-specific `Makefile`, as well as the files of the application to be tested. When no absolute path is given, the path is interpreted relative to the setup folder. The generated

scripts however will contain absolute paths to make them more robust against the environment they are called from.

The lines 9 to 12 trigger the execution of `make`. The `addConf` function is C-Specific and allows setting C-macros to different values. In this case the macro `CCA_THRESH` is set from −54 to −20 in steps of 2 and the macro `TEST_RATE` fron 5 to 30 in steps of 5. The generated code assumes that the environment variable `CLFAGS` is passed to the compiler, as is common practice. The Make-class will therefore generate shell-script code that looks similar to the following:

```
CFLAGS="-DCCA_THRESH=-50 -DTEST_RATE=10" make
```

The next step, line 14, executes the Cooja simulator with `test1.csc` as simulation configuration. As the file path is not absolute, it will be taken from the setup directory, but referenced using an absolute path in the generated script. As the simulation is a Monte-Carlo simulation, it must be run multiple times, being initialized with different random seeds. In the example the seeds 1 to 50 are used, creating 50 experiments for each configuration. Assuming that the Pseudo Random Number Generator (PRNG) is sufficiently good, it does not matter which seeds are used as input.

With the call to `generate` (line 18) the campaign folder with 36 000 different experiments is generated (45 `CCA_THRESH` × 16 `TEST_RATE` × 50 random seed). For each experiment a script is generated that contains all instructions necessary to set up the environment in the execution folder, execute the experiment and collect the results. The files to be recovered from the experiment environment are either automatically added by the classes provided or can be manually added to the experiment setup. The generated shell script also contains basic error handling, as it is not reasonable to start a simulation if the compilation fails.

5.1.1 Provided Functionality

In the following I will give an overview of the steps that can be added to an experiment. A full documentation can be generated using the source code.

Setting up the Environment There are multiple classes that help to set up an experiment environment.

- **GetDir** recursively copies a folder to the experiment environment.
- **GetFile** copies files to the experiment environment. It is possible to use wildcards as these are passed to the test script, which is executed by a shell interpreter.
- **Link** creates a symlink in the experiment environment.
- **Git** allows exporting revisions from a git repository. The revisions to test can be added explicitly using its hash, by setting a range using two hashes, or a hash and a number of preceding versions. All techniques can be combined. It is even possible to only add commits from "left" branches. This allows excluding so-called feature-branches from the experiments.
- **RealSim** supports the handling of RealSim files. It allows setting a range and a length to take from a RealSim file. This range is than evenly distributed to the different random seeds the Cooja is run with.
- **Clean** removes a file or folder from the target environment. This is for example handy when checking out files from Git.

Linking the RealSim trace to Cooja's random seeds was done to avoid adding another dimension to the parameter space. To get a reasonable coverage of a trace this would have required quite some experiments. Increasing the number of experiment by the factor of at least 24, which would, for example, represent 20 min of every hour using a 24 h trace, would have caused a simulation campaign to take 24 times as long. Instead of a few hours the campaign would take a few days to be executed. As trace and seed are truly independent, connecting seed and sub trace is statistically tolerable.

Running Experiments Once the environment is set up, the experiment can be run. The test generator does not distinguish between commands that set up the environment or execute a command. Thus it is possible to, for example, run a simulation, adjust the environment and run a second simulation. Files are however only recovered from the execution directory in a last step.

- **Make** runs the make command. It is possible to set C Marcos, which are propagated using the `CFLAGS` environment variable, add arbitrary commands to the `CFLAGS` environment or add parameters to be passed to the make command. Each configuration option creates a new dimension to the parameter space. Helper functions allow setting ranges, optionally with a step width, or lists of options to the configuration options. These can be freely combined, allowing to, for example, set 1 to 10 and 20.

- **Cooja** executes the Cooja simulator. It is possible to add random seeds to the configuration space, either be adding specific values or generating them using a PRNG. The PRNG is initialized with a user-defined seed to ensure recreating identical campaigns.

  At the time of the development, setting the random seed was not yet possible from the command line. Therefore, the simulation configuration is copied to the experiment environment and adjusted to reflect the selected seed.
- **Wisebed** controls the Wisebed testbed management software controlling the testbeds used in this theses. By integrating support for Wisebed into the test generator, it became possible to also generate test campaigns to be run on the testbed. As simulation and the real nodes in the testbed use the same binary image, the setup of the environment can be shared between simulation and testbed.

  To compensate for effects caused by a fixed node startup, it is possible to set a time frame within which the nodes are randomly started. These offsets are generated using a PRNG, the seed of which can be set.

5.1.2 Running Simulations in Parallel

One of the core advantages of simulation, opposed to running experiments in a testbed, is that it allows running multiple experiments in parallel. As each experiment is put into a self-contained shell script, any batch processing system can be used to run the experiments. In the evaluation GNU Parallel [Tan11] was used, which allows connecting to multiple computers using ssh, detecting the number of available cores and starting a corresponding number of jobs. It also provides file synchronization features, which however were not used, as all file exchanges were done using a network share.

5.1.3 Changes to Cooja

To allow running the simulations seamlessly, multiple changes had to be made to the Cooja Wireless Sensor Network (WSN) simulator.

- Cooja was not designed for normal plugins to do more than interact with the user. For this reason, when running without Graphical User Interface (GUI) it stopped loading plugins as soon as the script plugin was loaded. Thus, whether a plugin was loaded depended on the order of the plugins being referenced in the configuration file. With `becd2d4e` [Con] all plugins are always loaded, even when running without a GUI.
- Originally it was not possible to run Cooja outside its build path. The working directory had to be the same as the programs; and the logs were saved in the same folder, too. Therefore, it was not possible to execute the simulator twice without interfering with the log files unless Cooja was copied to separate folders. Multiple changes ware made to improve this: `64c0b9c2`, `32aa70e5`, `142fa4a9` [Con]
- Cooja crashed if the simulation ended without an output by the test script. This was triggered because the use of CoojaTrace, which will be presented in the following, made the use of test scripts superfluous. This was fixed in `50346251` [Con].
- It is possible to configure Cooja's logging using a Log4J config file [log]. This however was hard-coded into Cooja and could not be configured as a start-up parameter. When running experiments adjusting the verbosity of the log file is a convenient feature. Support to pass a configuration during the invocation was added with `3c3610d9` [Con].

5.2 CoojaTrace

CoojaTrace was developed in the context of the work under my supervision. [SLK12; Flo11; Cooa] It combines the Scala run-time compiler with a functional reactive programming framework. It allows to write programs similar to mathematical equations.

Listing 5.2 shows an example of the Energest approach [Dun+07] ported to CoojaTrace. The script is called when the simulator is initialized. It loops over all available nodes present in the simulation (line 1). Note that the naming scheme of nodes as motes in the code is taken from Cooja. The `timeSum` function is provided by the plugin and accumulates the time the condition is true. For example `mote.radio.receiverOn` is true, while the node's radio chip is in receive mode. The variables `rx`, `tx`, `act` and `idle` therefore contain the time the radio is sending or receiving, as well as the CPU being

```
for (mote <- sim.allMotes) {
    val rx = timeSum(mote.radio.receiverOn)
    val tx = timeSum(mote.radio.transmitting)
    val act = timeSum(mote.cpuMode === "active")
    val idle = timeSum(mote.cpuMode =!= "active")
    val energy: Signal[Double] = rx * 60 + tx * 53.1 + act * 5.4 + idle * 0.1635
    log(logt, mote, "energy", energy)
}
```

Listing 5.2: Logging the energy consumption using CoojaTrace

active or idle. The consumed energy is approximately the time spent in these states multiplied with the energy required in this state (line 6). Finally the data is logged to a log target (`logt`) saving the node, the string `"energy"` and the calculated value `energy`. Each time one of the conditions passed to `timeSum` is updated, all derived values are updated, too. The log target can write the data to a file, together with the current simulation time.

The plugin comes with helper functions to set up the logging functionality, but also provides a Scala-Interpreter within Cooja. It is possible to add arbitrary code to be run during the simulation and can therefore replace the test-script engine provided by Cooja.

CoojaTrace also has two extensions that allow using a SQLite[1] database [Cooc] or a graph as log target [Coob]. Using the latter it is possible to graph "live" data when running a simulation.

5.2.1 Wisebed Client

Wisebed [Cou+12] is the infrastructure used to manage the testbeds. Using Wisedbed to manage the local testbed had the advantage of also being able to access two additional testbeds in Brunswick and Lübeck. To interface the Wisebed testbed management software, a client was developed, similar to the test generator [Sca]. It combines the Scala run-time compiler and a library that allows programming the steps of an experiment, without having to cope with the low level interface provided by Wisebed. Adding the abstraction layer was necessary because Wisebed consists of multiple services that interact with each other and the interface works at a very low level. Logging into the system and reliably flashing a node requires several hundred lines of code. Most of them are needed to handle possible errors.

The client supports the following features:

- **Reservations** To interact with the nodes a reservation is required. This feature is provided by Wisebed to allow multiple users to use the same testbed without interfering with each other. When making a reservation, it is necessary to first check whether one already has a reservation for the nodes in question and if not, to make that reservation. It is also checked whether the remainder of an already existing reservation is long enough to run the experiment and if the necessary nodes are available.

- **Flashing** Multiple helper functions are provided to flash nodes. It is possible to flash all nodes with the same firmware, provide a map that assigns the nodes to a firmware and the combinations of both: If a node is not in the map it is programmed with the default firmware. All functions allow setting the number of retries. This was necessary because Wisebed failed quite regularly to flash nodes. The cause of this could be traced down to issues with the library used to interface the serial and the Linux kernel. I was however not able to find the exact cause.

- **Reset and startup** The client provides the ability not only to send the reset command to all nodes, but also doing so in a random order within a certain time frame. This allows minimizing the effect of boot-up order and inter-node offsets. It is also possible to block and wait for a certain string to ensure that the node successfully booted before progressing with the experiment.

- **Logging and serial communication** To interact with the nodes it is possible to send commands to the nodes' serial. Wisebed provides the data retrieved from the serial as raw data. Helper functions support splitting these up into lines. These can either be logged to one or multiple files, one for each node, or printed to the console. Of course a more sophisticated logger can be part of the script and interact with the nodes by sending them commands.

[1] `https://sqlite.org`

Independently of how many loggers are connected to a node, they all receive the same data. Thus it is possible to log the nodes' output to a file, print it to the console and interact with the nodes without loosing data.

- **Collecting results** Experiments are supposed to run for a certain duration. At the end the experiment can just be terminated. Sometimes it is however desirable to collect additional information from the nodes. For this a special function is provided to send a string to the nodes (e.g. `printstats`) and collect the answer. Of course the command also has a timeout in case one or all nodes fail.

5.3 Data Extractor

The raw data created by the experiments is saved as files in a sub folder of each experiment (see section 5.1, page 91). These typically contain the output of the serial or debug information provided while running the experiment. The relevant data must often be extracted or derived from the information provided. The data must also be put into correlation to the input parameters. To ease the data collection and correlation with the input parameters, the data extractor framework was created [Dat].

The framework provides multiple pluggable filters that extract the information from the files generated by the experiment. These pass the extracted information back to the framework as key-value or node-key-value tuples. The returned tuples are merged with the key-value mapping of the experiment's configuration. A comma-separated file format is used as output. This allows reading the data into most other applications, for example R, for further processing with minimal effort.

Except for very simple filters, for example to extract the size of the different memory sections or extract the number of reboots from the simulator's log, the filters are test case specific. To support the development of new filters, the framework already provides multiple helper functions that, depending on the file format, strip unnecessary data or extract the node ID. The full documentation is provided as part of the framework.

6 Evaluation

The evaluation will concentrate on the feasibility of the approach chosen in this thesis, as there is no published work, or at least no work related closely enough, to allow direct comparison. One way of evaluating the approach would be to use the simulator to find configurations that perform better than the default configuration and verify the simulation's result using real hardware. As I will discuss below, this strategy is likely to yield a false positive results. Instead, I will try to answer the crucial question: How close does the simulation come to the real world and is this sufficient to optimize parameters?

It is important to keep in mind that the results gained from a WSN have a high amount of jitter and the network can only be simulated, but not emulated. The limitations of simulating WSNs have been discussed in section 3.6, page 52. Due to their inaccuracy, simulations can only provide trends and estimations, but no accurate results. Thus, simulation and testbed can only be compared qualitatively, but not quantitatively.

When evaluating the presented approach by finding a superior configuration using simulation and then verifying the superiority using a testbed, only two data points are provided: The original configuration and the new, optimized configuration. As the comparison can only be done qualitatively, the evaluation is positive when the real hardware yields better results using the allegedly optimal configuration. It is however not necessary that it is also the most suitable configuration as suggested by the simulation.

For example, it might be that the default size of a buffer is 3. The simulation found that any value above 10 yields optimal results in terms of performance. In reality this could also have been achieved with a value of 5, and therefore a lower memory footprint. Thus, when only looking at these two values, the evaluation is positive, yet missed the target of finding an optimal value, which would have been 5.

Consequently, a sound evaluation can only be executed, when it is also proven that the found solution is also the optimum for the real network. This however requires searching the same parameter space as was done using simulation. This requires about the same effort as answering the more extensive question of how realistic the simulation is.

While the main technical contribution of this work is mapping a real network accurately to a simulator, the whole approach heavily relies on the tools used, specifically the simulator. The simulator used has quite a few short-comings and bugs, some of which have been removed as part of this work. Nonetheless, the simulator is a core component of this work and the approach cannot be evaluated without taking it into account. Verifying that the data was correctly acquired and fed into the simulator, as well as the necessary bug-fixes were therefore already discussed in chapter 3 (Analysis, page 39) and chapter 4 (RealSim, page 79). Part of this research was therefore evaluating the state-of-the-art of realism in WSN simulation.

To answer the question, whether the simulation is accurate enough, I took the following approach: I created multiple campaigns covering a certain parameter space. The parameter space was explored using the simulation, as well as the testbed. The results of testbed and simulation where then compared and interpreted. Technically the quality of RealSim (and the simulator) was evaluated using DryRun as test harness. By taking this approach, it is possible to compare the impact of changing configuration options on the simulator and the testbed.

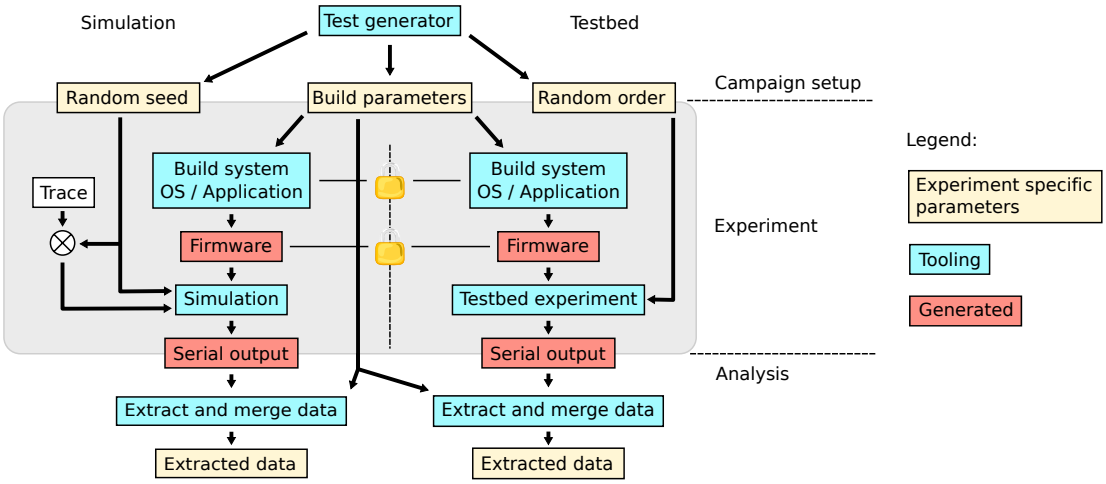


Figure 6.1: Conceptional structure of the experiment campaigns used for the evaluation. The input for the experiments is generated using the DryRun's test generator. The test generator also generates the script that executes the experiment. As the testbed and the simulation experiments share build parameters and build system, identical firmware files are generated.

Figure 6.1 gives a conceptional overview of the structure of the campaigns used for the evaluation. Each Campaign is split into three parts: Campaign setup, experiment and data analysis.

The DryRun test generator is used to create two similar campaigns that only differ in the environment specific attributes (for example, for the simulation a trace of the testbed is needed). Each campaign consists of many experiments, with each experiment having different input parameters. To account for random effects, multiple experiments have the same build parameters, but differ in their random seed or the time at which they are executed in the testbed. The DryRun data extractor merges the output with the input parameters. When comparing the data of simulation and testbed, the experiment itself is a black box, as only input and output parameters are known.

To allow comparing simulation results with those from a real testbed the same input and output parameters must be accessible for both systems. For this reason I chose to use parameters passed to the build system as input. As the build system and the source code built for the simulations and the testbed do not differ, the resulting firmware images are identical, too.

The output data must also be acquired form simulation and testbed in the same way. The only interface to the nodes that can be accessed from the simulation and the testbed is the serial interface. This however means that the data must be collected by the software running on the nodes and then printed to the serial. To run the experiments, it is therefore not possible to run any software. Instead the software must collect relevant statistical data and provide these on demand. Fortunately Contiki-OS already has built-in functions to collect statistics about the network layer, as well as the time spent in certain states. Of special interest are the durations the CPU is active or in sleep mode, as well as the radio receiving or sending. Based on this information it is possible to estimate the energy consumption (see section 3.6.10, page 61). Extending existing software can therefore be done with minimal effort.

As discussed in section 3.6.1, page 53, the simulation has a (pseudo-)random component. To get statistically valid results, the experiments must be repeated using different seeds for Cooja's PRNG. The same applies to the testbed: The experiments must be run multiple times to compensate random effects.

Another issue are the changing conditions. As all testbeds are in an office environment the conditions change throughout the day and even at night they are not necessarily stable. For the simulations this is taken into account by using a longer trace and assigning different snippets to each experiment, which correlate to the random seed.

For the testbed this is not as simple. Providing equal conditions for different configurations is not easy. When running them back-to-back it is likely that two experiments have similar conditions. However, due to the sheer number of different configurations it can easily take several hours until each configuration is run once. Thus the first and the last configuration are very likely to face varied environmental conditions. To address this, the experiments of a campaign are executed in a random order. This is not optimal as it means that there is a theoretical chance that all runs of one configuration might run under good conditions, and all runs of another configuration run under bad conditions, yet at least statistically all configurations are treated equally.

Finally, after running simulations and real-world experiments, it is possible to compare the results of simulation and real-world experiments based on the input parameters.

Software and Parameters The software used for the evaluation must meet certain requirements. As already noted, it must provide means of data collection and printing via the serial, as this is the only means of data collection that is provided by real hardware and simulation. The data collected should of course reflect areas of interest, as should the input parameters. As discussed, the challenges in WSN simulation are mainly related to the network domain, as most hardware can be accurately simulated. Consequently the focus will be on the network, rather than Operating System (OS) or application related topics.

These aspects can however be distributed over the different layers of the system. The network is obviously influenced by the network stack, but also by scheduling decisions of the OS, as well as the rate at which the application sends its data to other nodes. In the evaluation this is taken into account and it is tried to cover a large range of features, specifically those that have an influence on the radio chip. Notwithstanding the tools provided, specifically DryRun, can of course also be used to tune parameters that are not related to the network.

Something that is of minor importance for the validity of the evaluation is the correct working of the software. As the simulation is supposed to reflect the behavior of the real testbed, bugs should show systematical errors in the data of both, the simulator as well as the real testbed. If, for example, there is a bug when accounting the number of packets and all packets are counted twice, this should be visible in simulation and testbed. Ideally the number of packets being sent is the same in both environments and therefore the returning numbers are the same, too, yet twice as high as the number of actually sent packets. If this is not the case, this is a problem of the simulation environment not accurately reflecting the real world, rather than the software. In the end, the goal of the evaluation is not to find an ideal configuration for the tested environment, but to show that the accuracy of the simulation is high enough to generally achieve optimizing WSN software.

6.1 Execution of Experiments

For each experiment campaign a trace of at least 24 h was taken. Afterwards the experiments were run on the nodes. As such an experiment campaign often took over a week, it is not unlikely that conditions changed significantly, causing the simulated environment to differ from those of the experiments. Unfortunately there is little that can be done about this. As the trace represents the approximate state before the execution of the experiment campaign, no time constraints are bound to the simulation. In fact many simulation runs were run months later to see how certain changes to the simulator or its model affect the results.

During the experiment statistics about the network stack and times spent in certain system states were collected. This was done using the monitoring functionality provided by the OS. After receiving a special command, the application running on the node retrieved this information from the OS and printed it to the serial port. Depending on the experiment, additional information, for example about sent and received packets, was printed to the serial as well.

All campaigns were generated using the DryRun test generator, as described in section 5.1, page 91. It generates the scripts that setup the build environment, build the firmware, execute the experiment and collect the results. Both, the scripts to run the simulations and the campaign on the testbed were derived from the same test-generator script. A switch was used to either trigger the invocation of the simulator or testbed management client. By doing so, it was ensured that the build environment, and therefore the firmware, are identical. The configurations used to set up an experiment campaign are described in the appropriate sections.

6.1.1 Simulation

In all campaigns the sink node had a special firmware that differed from the other nodes. To ensure that the mapping of firmware to node id is correct, all nodes were pre-created as part of the simulation environment setup. Although RealSim supports the creation of nodes, even of a specific node type, this was not used for the experiments. RealSim has no support to create node types, the combination of hardware and firmware file. These therefore had to be created manually. The overhead of also creating instances of the nodes was so marginal that it did not justify making the necessary extensions to the DryRun tool chain to also create nodes of the right node type.

By default, Cooja does not write the serial output of the nodes to a log-file. Cooja however provides the ability to run scripts for automated testing. This was used to write the serial output, tagged with the node id, to a separate file.

The test script was also used to send the commands to the nodes to print the statistical data at the end of the experiment. After the collection of all results from the nodes, the simulation was terminated.

6.1.2 Testbed

When running an experiment using the testbed, a special client that is described in section 5.2.1, page 95 was invoked instead of the simulator. It was responsible for all the steps required to run the experiment in the testbed.

After flashing the nodes, a command was sent to the testbed management clients to reset the nodes. The commands were randomly dispatched over a period of 3 s to allow random inter-node offsets as discussed in section 3.6.2, page 54. Every line printed to the serial during an experiment was tagged with the node it originated from and a timestamp, and was written to a log file.

As in the simulation, a special command was issued at the end of an experiment to collect the statistics using the serial interface. Afterwards the experiment was terminated.

6.2 Data Analysis

During the experiments three types of data were collected: Network statistics, system state timings, which can be used to derive energy statistics as discussed in section 3.6.10, page 61, and information generated by the application to be evaluated. The latter normally contained the dispatchment and reception of packets at the application layer.

The amount of raw data collected does become quite large. By default 4 different system states – sleep mode, active, listening/receiving and sending – and about 50 different network attributes are collected. When running 2000 experiments per campaign this sums up to over 100 000 different values – for each node. While the network attributes are interesting for close analysis and understanding of why the system behaves as it does, this is of minor interest when making a general comparison between simulation and real world. Instead, I will concentrate on two main attributes that are of general interest: The energy consumption and the number of packets sent by the application that successfully reach the sink.

I also aggregated the data for all nodes taking part in the experiment for most campaigns. This was done to reduce the hight amount of jitter between multiple runs of the same experiment[1]. Inter-node offsets and random packet loss, as discussed in chapter 3, page 39, can decide whether one or the other node ends up as the default route. Thus, even in the same environment, all packets can be routed via one or the other node. At the node level this will have a huge impact. If the two nodes have similar connectivity, this is likely not visible at the network level.

6.3 Analysis of a Trace

Aggregating background noise and its limitations were already analyzed in section 3.7, page 63. It was found that to accurately reflect the background noise a model is needed that does not yet exist. The collection of packet attributes like the Receive Signal Strength Indicator (RSSI) and Link Quality Indication (LQI) has been discussed theoretically in section 3.8, page 73. The assumption was that RSSI and LQI of a connection do not change much within a small time frame and it is therefore possible to aggregate them using an average. In the following I will therefore look into whether this assumption holds and also generally discuss the data collected.

For the analysis a ≈24 h trace was used. To get a better understanding about the data, the RealSim network tracing software was extended to not only print the aggregated data, but also the raw values collected from the radio chip. This was only possible, because the data was not forwarded to the sink using the network, but directly printed to the serial.

For the traces used in the evaluation 8 packets were sent per 80 s epoch. This was an arbitrary value, which was chosen to reduce the chance of collisions at a time where the testbed in Lübeck, which had a high number of closely meshed nodes, was used for experiments. After having to yield using the testbed in Lübeck, the number of packets per epoch was not altered. Analysis of the simulations however showed

[1] This applies to testbed and simulation. The latter using different snippets of the trace and seeds for the PRNG.

that other effects, specifically background noise, insufficiencies in the network model and random effects had a stronger impact than the low number of packets used to estimate the attributes of the connection.

6.3.1 Packet Reception Ratio

Figure 6.2 shows the PRR of a ≈24 h trace. The x-axis displays the local time while the y-axis shows the PRR. Each dot represents the PRR of an epoch. As a result of using only 8 packets to estimate a connection, the PRR can only take 9 discrete values: 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875 and 1 This is also clearly visible in the graphs.

The connection characteristics between the nodes differ strongly: Nodes 2 and 3, 6 and 9, as well as 5 and 10 are placed close to each other, as can also be seen on the map of the testbed (fig. 6.8, page 108), and therefore have little packet loss, though not none. Interestingly also the connection between nodes 8 and 9 is very good, although they were placed several rooms apart. This clearly shows that the connection quality does not only depend on the distance and obstacles between nodes. Why these nodes had a good connectivity despite the distance is unclear.

Changes The graphs show changes over time. Nodes 1 and 6 have an average connection at the beginning of the trace, which then degrades. At the middle of the trace the connection improves to a good connection, to degrade again towards the end of the trace. This might be attributed to changes in temperature, as there is no activity during the night, yet the rooms cool down. Node 5 seems to have the same issue as node 1 when communicating to node 6, as the connection degrades at the beginning of the trace. Yet the connection does not recover. Nodes 6 and 8 on the other hand have relatively good connections that degrades for two hours, between 14:00 and 16:00, to recover afterwards. The cause might have been a regular meeting that happens regularly in one of the offices that are crossed by the connection. The graphs show too many features to be discussed entirely, yet the core statement that connections are subject to, possibly very strong, changes caused by the environment is apparent.

Asymmetry Connections do not only change over time, but they are sometimes also asymmetric. While the very stable long-range connection between nodes 8 and 9 seems symmetric at first sight, this is actually not the case. Very few packets get lost from node 8 to 9, while it happens quite often that one of the packets gets lost in the opposite direction. The nodes 1 and 4 also have an asymmetric connection: While the characteristics of the packet loss from node 4 to 1 do not change much over time, it is visible that the amount of packets lost in the other direction changes over time.

The connection between nodes 5 and 9 also shows interesting aspects. Around 14:00 the connection is more or less symmetric. Yet after 20:00 the connection becomes very asymmetric with a very good connection from node 9 to 5 and a rather bad connection the other way round. During the night, and clearly visible after 2:00, both connections degrade. While the cause of the change around 20:00 can probably be traced back to doors being closed after people leaving their office, it can only be speculated about the degradation during the night. Most likely this is caused by changes in temperature as there are few other attributes that gradually change during the night.

Timeframe Figure 6.3 shows two ≈24 h traces that were originally taken right after each other to compare the effect of using the default number of packets/epoch (8) and one using 32. The trace only shows the connectivity between the nodes 1, 3 and 4. Only node 1, the sink node, is connected to other nodes. Therefore, these three nodes cannot communicate with each other using other nodes.

Nodes 3 and 4 are placed close together, and therefore have good connectivity in both directions. Despite there proximity, only node 3 can communicate with node 1. One can see that the connectivity between nodes 1 and 3 was better while the trace with the low resolution was taken. The low-resolution trace always has average to good connectivity, while the high-resolution trace has average connectivity during the night and almost none during the day.

As will be shown in the campaign presented in section 6.7, page 118, the trace with the low resolution will yield simulation results that are closer to the experiment than the trace with the high resolution. This is not due to the high resolution yielding worse results in general, but due to the trace with the low resolution representing a time frame that is more typical for the network. This shows that the time frame in which the trace was taken has a higher impact than increasing the resolution of the PRR to more than 9 discrete values.

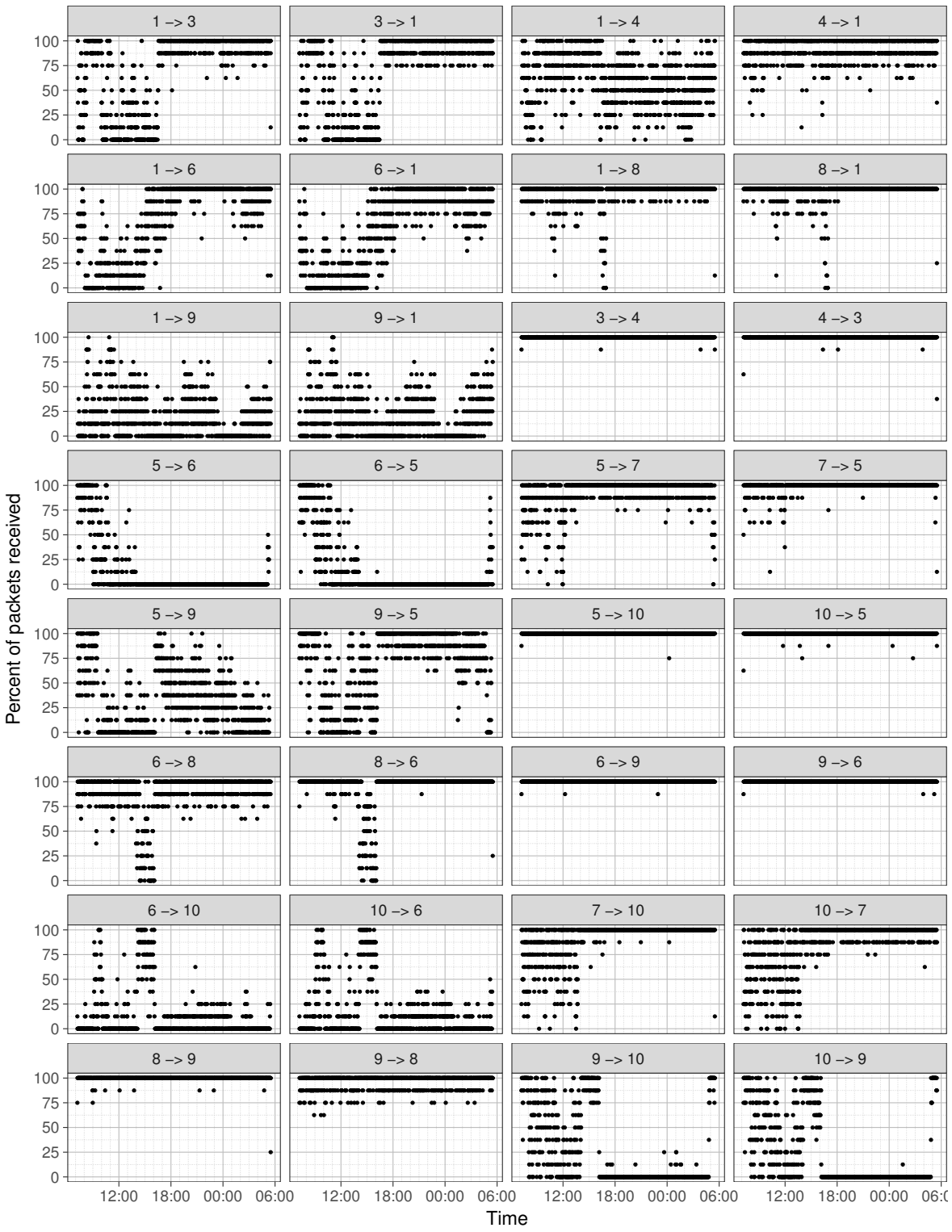


Figure 6.2: The PRR collected over a duration of $\approx 24\,\mathrm{h}$ at Erlangen. Each point represents the data collected during an epoch of $80\,\mathrm{s}$. Correlating connections are placed next to each other, the graphs are sorted by the sending node. Data: [Str17a]

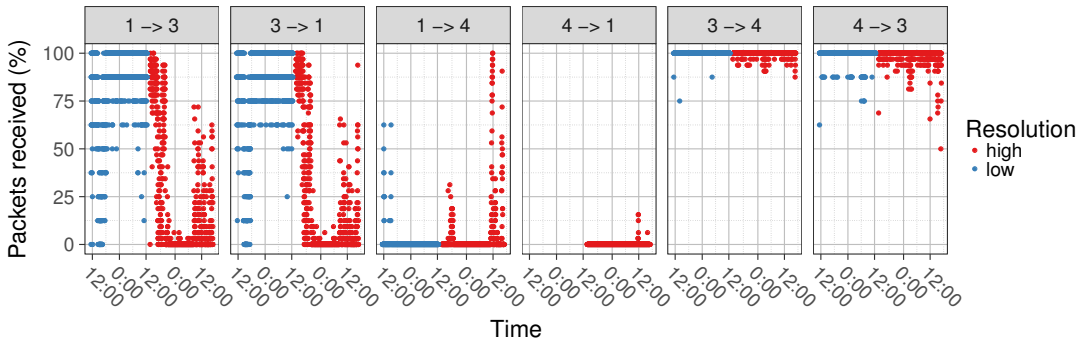
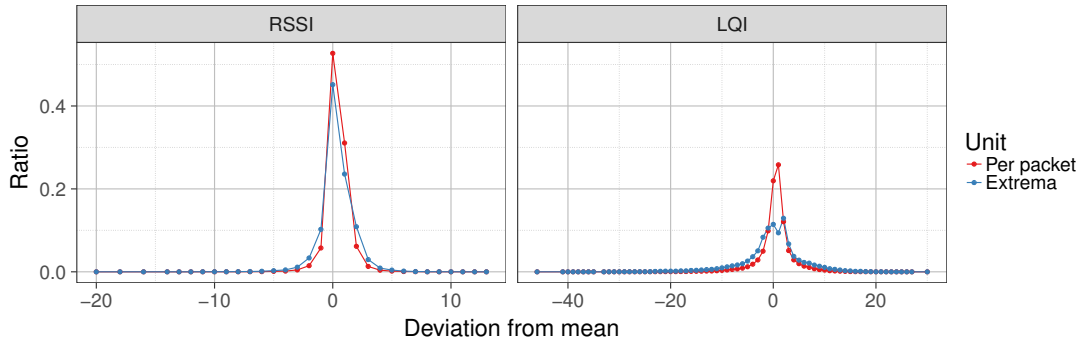
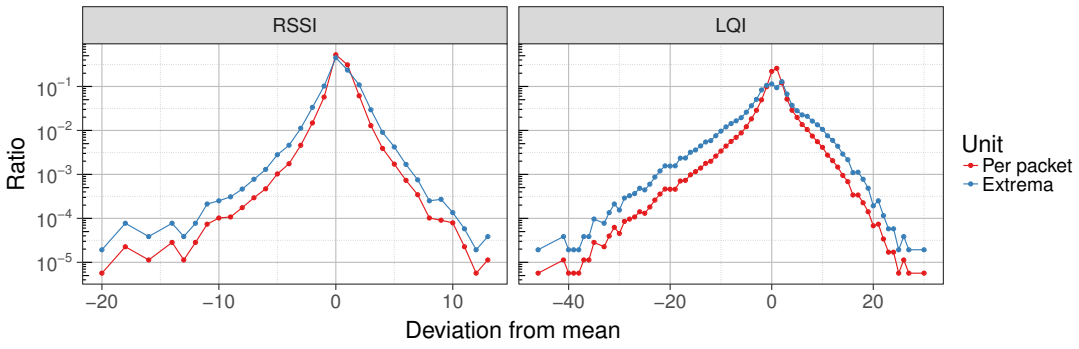


Figure 6.3: Two traces taken back to back. One at the default resolution using $8\,\mathrm{packets/epoch}$ and one using a higher resolution using $32\,\mathrm{packets/epoch}$. The fist trace was started around 11:00 in the morning, while the second was started at 13:00. Both ran for about $26\,\mathrm{h}$.



a) Linear scale



b) Logarithmic scale

Figure 6.4: The deviation of the read values to the averaged value base on $176\,548$ samples collected over 32 connections and 1007 epochs, resulting in 5.48 samples per epoch. The red line represents the ratio of the deviation calculated over all packets, while the blueish line only takes the extrema into account. Data: [Str17a]

6.3.2 RSSI and LQI

To see how strong the RSSI and LQI values change during the epoch of 80 s, the raw values were compared to the calculated mean for every epoch. The data is shown in fig. 6.4. The x-axis shows the offset from the mean calculated by the software on the node and the packet for each epoch. The y-axis accounts for how many packets have this offset. The red line shows the deviation from the mean for all packets, while the green line only represents the packets with the maximum and minimum offset per epoch.

That the lines look similar, but do not converge towards the extrema, is due to the y-axis representing a ratio. There is an average of 5.48 packets per epoch, but the extrema are only represented by 2 samples. The lines towards large offset represent the same absolute numbers, yet as there are more samples than extrema, the ratio of the samples is lower. There is also a clear bias to $+1$, opposed to -1. This is caused by the averaging algorithm rounding off, resulting in a too low average.[2]The effect is especially strong in a low noise environment: If there are 7 samples of -90 and one sample of -89, the calculated average will still be -89. I will come back to this when looking at the same data broken down by connection.

The graphs in fig. 6.4 show that the vast majority of the samples taken within an epoch do not differ much. For the RSSI only 10 % of the packets and 20 % of the extrema differ more than one from the calculated average per epoch. Small changes can be expected, as the signal strength should not change much, as long as the environment does not change much either. The strong changes of over 10 dB can however only be explained with significant changes in the environment, for example a door being closed, or bugs in the hardware as discussed in section 3.7, page 63. Considering the overall inaccuracies, the changes in RSSI during an epoch can be considered tolerable for the time being.

For the LQI things are a bit different. Here the changes are stronger, which can be seen at the much lower peak at the low offsets as well as the extreme offsets being much higher. Also, 42 % of the packets and 69 % of the extrema differ more than one from the calculated average. Again this is to be expected: Both, LQI and the RSSI values presented are only collected for successfully received packets. A packet can only be received, when interfering signals were weaker than the received signal. As RSSI is proportional to the incoming signal power in dB, the weaker signals can only have a small impact on the measured value. The LQI on the other hand is an arcane value that is relative to the quality of the received signal. Thus, a weaker signal that does not influence the RSSI can have an impact on the LQI.

There is also a dent for packets that deviate by $+1$. This is caused by a combination of the suboptimal averaging algorithm and the fact that the LQI values spread stronger than the RSSI values. The reason for the value at 0 being this high, is caused by epochs with only one packet being received. In that case that one packet will yield two extrema that both do not deviate from the mean.

Figure 6.5 further breaks the data down by the receiving node and connection. Deviations above ± 5 were cut off, as these are close to 0. Also a linear scale is used, to improve the visibility of the differences between the datasets.

Looking at the offsets of the RSSI values, the nodes can be split into two groups: Those where slightly more than 50 % and those where slightly less than 50 % of the samples deviate from the calculated average. The latter have a high number of offsets by $+1$, which is caused, as discussed, by the averaging algorithm. Despite the distinctiveness of these two groups there is no correlation to the nodes, neither in terms of location, nor hardware design[3].

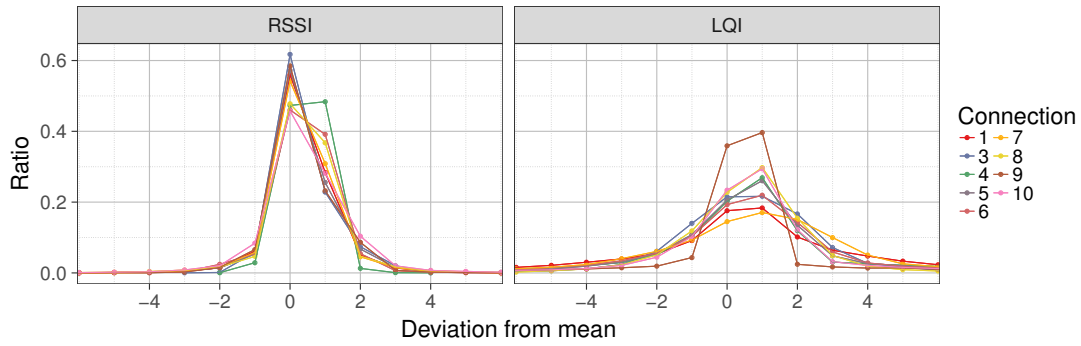
The LQI is very similar for all nodes, with exception of node 9. For node 6, which is near node 9 and has the same connections to other nodes, the variation of the LQI is much higher. This is most likely caused by variances in the hardware, as noise should rather influence the stability of the LQI negatively, while node 9 stands out positively.

The RSSI values broken down by connection show that there are some variances between the connections in terms of stability. Besides some shifts from selected 0 to $+1$, as well as from $+1$ to $+2$, which can be explained with the averaging algorithm, they are all very similar.

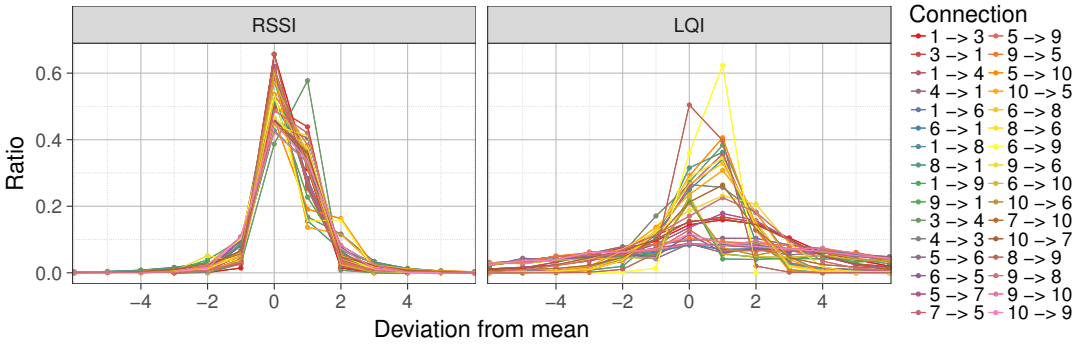
This does not apply to the LQI values. Here the connections are more distinctive. Some of them have very distinct values, while other seem to have a strong variance. Investigating this further might be interesting to get a better understanding of channel properties. Yet the LQI has no impact on the simulation model, nor the network protocols used in this evaluation. I will therefore not look into this further.

Averaging To better estimate the effect of the averaging algorithm, the data was processed to reflect a rounding averaging function. The result is shown in fig. 6.6. All graphs are much more symmetrical, and the datasets have a more similar course. Besides yielding nicer graphs the effect on the accuracy of

[2]Internally the range of values is shifted to positive values. Thus, even negative values are rounded off, rather than up.
[3]The nodes used had identical schematics, yet differed slightly in their Printed Circuit Board (PCB) layout.

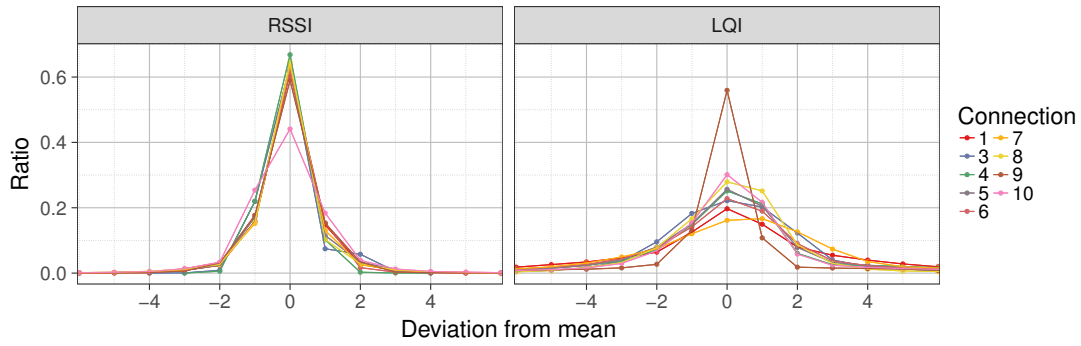


a) By receiving node

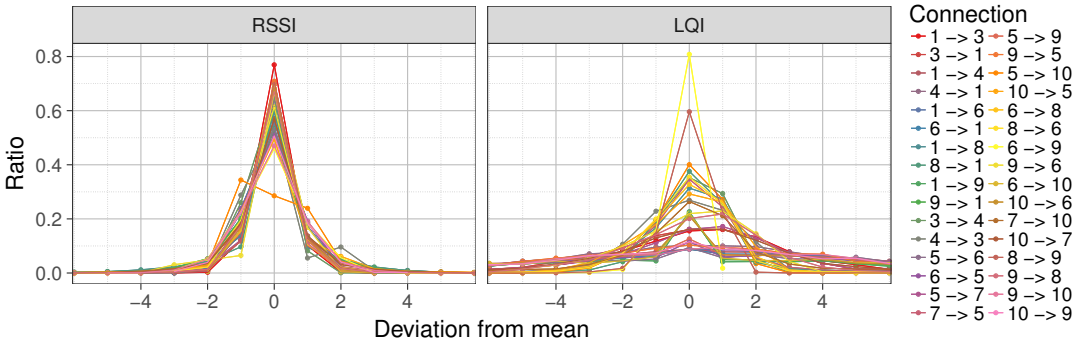


b) By connection

Figure 6.5: The same data as in fig. 6.4, yet broken down by receiving node or connection. To actually see the differences, the range was limited and linear scale was used. Data: [Str17a]



a) By receiving node



b) By connection

Figure 6.6: The same data as in fig. 6.5, yet with using a rounding averaging function. Data: [Str17a]

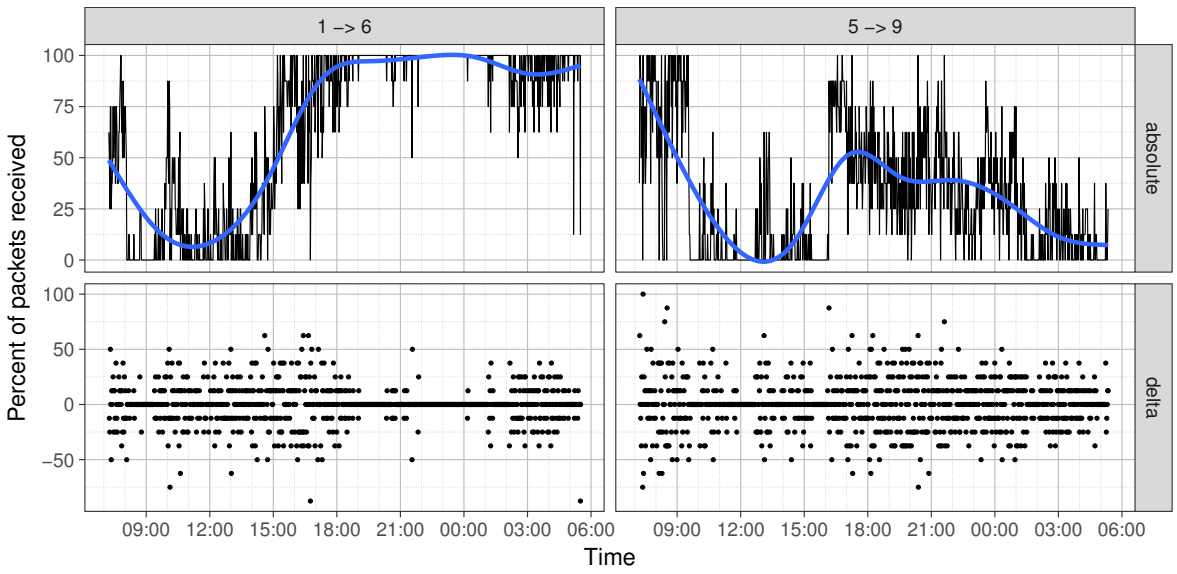


Figure 6.7: Instead of dotes as in fig. 6.2, page 102 lines are used to highlight the jitter. Additionally a smoothing function is used to visualize the long-term trend. The delta-graphs show the offset to the previous value. Data: [Str17a]

the simulation is marginal. After all, the averages are shifted by +1 at the most, affecting ≈44 % of the samples.

Discussion The evaluation of the data has some shortcomings. For example, when only one packet was received during a period, this one packet has the same value as the average. Also, this one value also represents the maximum as well as the minimum. A very bad connection will therefore yield only little variance. This is however a general problem when not having the same number of samples per aggregated subset.

Conclusion In conclusion, even though the averaging is not optimal, the effect can be considered tolerable. The main noise of the RSSI is limited to 2 bit (−1 to 2). Consequently, parameters that are sensitive to such small changes can currently not be tuned without additional real world experiments. For the testbeds used in this work, this is however of minor importance, as random noise caused by the environment makes tuning at this accuracy unreasonable.

As will be shown in section 6.5, page 110, from the investigated protocols, only ContikiMAC is affected by this inaccuracy at all. For the other two Medium Access Control (MAC) protocols that are provided by Contiki the actual strength of the received signal plays a subordinate role.

6.3.3 Discussion

The analysis of the trace shows that averaging RSSI and LQI over an 80 s time frame is unproblematic. And even the low resolution of the PRR using only 8 packets gets shadowed by the long-term (>24 h) changes of the network. All experiments were run based on ≈24 h traces, which, considering the results, met the default case quite well. Solely in one campaign (section 6.7, page 118) the results were unsatisfying and the consequences will be discussed there.

In the end volatile environments like office environments must be monitored for longer durations than 24 h to provide accurate predictions using simulation. This is one of the general problems with the current state of the approach and has already been discussed in section 3.8.3, page 74. The alternative is to use a short trace, for example 24 h and monitor the network while it is in production. An updated configuration can then be deployed, if necessary. The general approach how this can be done has been discussed in section 3.8.5, page 77.

One of the ideas in section 3.8.3, page 74 is to use the trace to model the environment. Such a model can be supported by taking a closer look into the source of the changes in connectivity. For example the connection between nodes 1 and 6 gradually degrades during the afternoon. As node 1 is positioned beneath a glass roof, this could easily be temperature related. Finding such correlations however requires collecting more information like temperature and ideally the state of the interconnecting doors, and room

occupancy. This however is rather the domain of WSN radio-network modeling and therefore out of the scope of this work.

6.3.4 Conclusion

The two proclaimed issues, asymmetry and changes over time, can only be solved by using traces. While it might be possible to predict the asymmetry, this is almost impossible with changes over time. Even though, the trace is only a snapshot of the WSN's environment and there is no guarantee that the time frame it covers resembles the typical state of the network. However, predicting the future of a WSN goes beyond the scope of this work and requires more research.[4]

6.4 Experiment Setup and Environment

In the following I will describe the environments and the setup used for the experiments. Deviations from the default setup will be explicitly mentioned in the descriptions of the campaign.

6.4.1 Software revisions

For the experiments the following software revisions were used. They are only noted as far as they are relevant and, to the best of my knowledge, are able to influence the results.

Contiki Contiki was used in revision `fe0a0423` [Con] (2013-08-11 16:14:28). As discussed in the introduction of this chapter, the actual version of Contiki should only influence the evaluation insofar, as the changes to another version interact with shortcomings of the simulation model or bugs in the simulator. All other changes might influence the actual numbers, but should not impact the result. The changes made to this version are as follows:

- The random seed was initialized with the same value on all nodes. By instead using the node id, it was ensured that the numbers generated by the PRNG differed, if not between experiments, then between nodes.
- It was not possible to change the duration of an event timer without also resetting the reference point. This made accurate time keeping using changing intervals impossible. A function was added to allow changing the duration of an event timer without resetting the reference. This change was accepted upstream: `9b77aac5` [Con]
- A check was added to detect when the radio chip is unable to create a valid RSSI value. This however was never triggered in the experiments.
- In the default configuration it was not possible to use Internet Protocol version 6 over Low power Wireless Personal Area Network (6LoWPAN) and the serial from the application layer at the same time. Access to the serial interface form the application layer was however necessary to transfer the system statistics and log information. The exclusion was removed.
- The Energest subsystem, which is responsible for the accounting of the system states, was initialized after the radio chip. However, during the initialization Energest has no knowledge of the state of the monitored subsystem. Thus accounting starts with the first state change. Under certain circumstances the radio state never changed after the initialization. For example when the radio listens without any Radio Duty Cycling (RDC) mechanism and never sends a packet. By initializing Energest before the radio, the initialization of the radio triggered a state change. The change was accepted upstream: `c0783e28` [Con]
- It was not possible to set the Clear Channel Assessment (CCA) threshold at compile time. While this was solved at the application level for the evaluation, a patch was accepted upstream: `c67c048c` [Con]

C-Compiler and Tool chain To create the firmware for the micro controller gcc-Version "4.6.3 20120301 (mspgcc LTS 20120406 unpatched)" was used.

Cooja The experiments were simulated using Cooja `b78eb96a` [Con] (2014-03-18 12:32:56) with the changes described in chapter 4, page 79.

[4] And most likely a crystal ball.

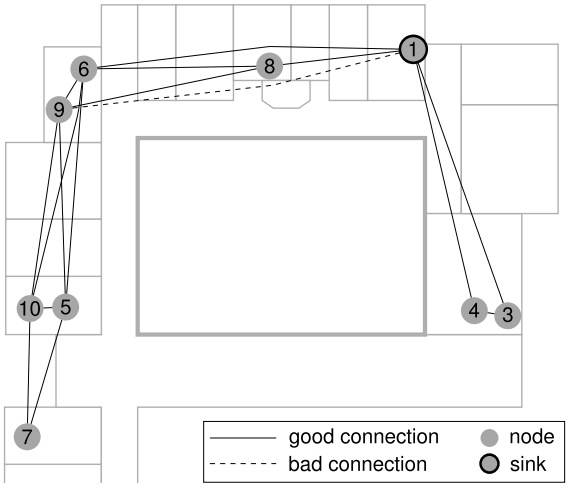


Figure 6.8: Layout of the WSN testbed in Erlangen. The room in the center is a lecture hall with thick walls that block the radio signal.

Other Tools Most tools, like those used to manage or interface the testbed, pre-process, evaluate and visualize the collected data should not have any impact on the results and are therefore not explicitly referenced. The other tools used are in compliance with their functional description in the according chapter.

6.4.2 Testbed

For the evaluation two testbeds were used, one in Erlangen and one in Brunswick, which were located in office environments and were built using Sky WSN nodes and their clones, which are presented in section 2.1.1.5, page 15. Both testbeds were managed using Wisebed [Cou+12].

During the initial development an additional, large, 52 node testbed in Lübeck was used. Yet a software update would have required significant changes to the client interfacing the testbed. Also, instability issues of the testbed infrastructure that were caused by frequently flashing the nodes were more severe due to the higher number of nodes. Opposed to Erlangen and Brunswick, I did not have direct access to the system and had to rely on the local administrators to recover the system. For these reasons this testbed was not used for the final evaluation.

6.4.2.1 Erlangen

The testbed in Erlangen was placed in the offices of the Department Distributed Systems and Operating Systems and is shown in fig. 6.8. The 9 Sky nodes were placed as far apart as possible, while still providing stable connections. They were also often placed in pairs, as this reduced the number of computers required to manage them and provided redundant connections, which, to my experience, lead to a significantly reduced number of lost packets. This was especially relevant when developing functionality to not only print the network survey data to the serial, but also send it to a sink node.

The room in the middle is a lecture hall with stronger walls, blocking the connectivity. The good connections, drawn as a continuous line, typically had a PRR of 100 % The connection between nodes 9 and 1, which had strong fluctuations and often had a PRR of 10 %, probably due to doors being open or shut, is marked with a dashed line. To make the connection more reliable, node 8 was added. Sometimes it was also possible to see sporadic packet receptions of nodes not shown as connected, yet these were rare.

The short ranged connections have a RSSI value of about $-40\,\mathrm{dB}$ while the long range connections were at around $(-85 \pm 5)\,\mathrm{dB}$. Especially nodes 3 and 4 were placed close to a Wireless Local Area Network (WLAN) access point, yet in a lab rather than an office. Thus, they were not as exposed to other devices like mobile phones that connected to the WLAN as the other nodes, which were placed in offices. For all experiments node 1 was used as sink node, which is marked with a thick border.

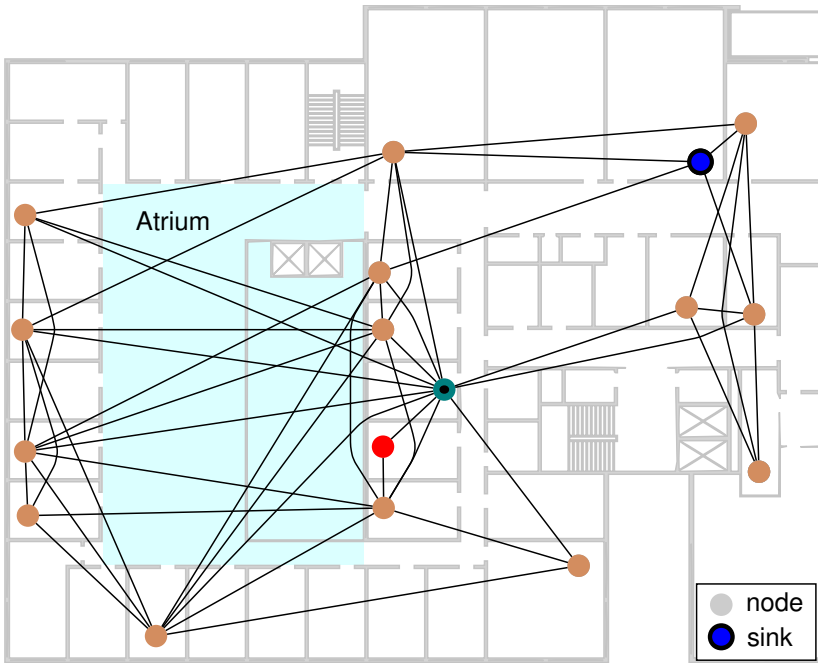


Figure 6.9: Layout of the WSN testbed in Brunswick. The highlighted area in the center is an atrium, providing good long-range connectivity. The red dot marks a node with very bad connectivity, while the greenish node to the top right of it is well connected.

6.4.2.2 Brunswick

The testbed in Brunswick was located in an office environment at the Institute of Operating Systems and Computer Networks and consisted of 17 Sky nodes. Figure 6.9 shows the good connections between the nodes. Inferior connections were not added for the sake of clarity. Most of the nodes ware placed in offices around an atrium, which is shown in a bluish-green shade. The atrium provides the nodes with a relatively good long range connectivity. In comparison to Erlangen the network is therefore much closer meshed. To increase the number of packets that must be routed, a node outside the central mesh was chosen as sink for all experiments. It is marked with a blue filling and a black border.

The figure also shows that, especially in an office environment, the distance is not necessarily related to the connectivity. In the center a green node marked with black dot has extraordinary connectivity in all directions, while the node right next to it, to the lower left, is only connected to two nodes. Similarly the node to the top right of the sink has better connectivity than the sink itself.

6.4.3 Application

The test application was derived from a simple 6LoWPAN User Datagram Protocol (UDP) test case where all nodes send data to a single sink, which then forwards the data to a gateway. It reflects the typical scenario of a sensing network. All nodes send the string `Hello %d from the client`, where `%d` is replaced by a counter, to the sink. For the result of the experiment the actual payload is only relevant insofar as it changes the length of the packet. The sending nodes also print the contents of the message to the serial. This was originally intended for easier debugging, but allows to easily count the number of packets sent and do a rough analysis of the timing. The sink prints the contents of every packet, annotated with the address of the sending node to the serial interface.

The rate at which the nodes send the data to the sink is configurable. The sending of data is implemented by having a timer that is triggered in regular intervals. To reduce effects caused by fixed offsets, the packets are not sent right away, but delayed by an additional random offset, based on the length of the interval.

As the sink node is connected to a gateway, energy consumption is considered to play a subordinate role. In the example code the experiments are based on, the sink's radio is therefore always kept on and no RDC is used. This was not changed for the experiments, yet the sink was excluded from any radio-time related analysis.

6.5 Campaign: CCA Threshold Versus RDC Protocol

The effect of background noise was already discussed in section 3.7, page 63. A parameter that strongly interacts with noise level, is the CCA threshold. By default, the radio chip is configured to compare the CCA threshold with the current RSSI value before sending a packet. If the RSSI value is higher, the channel is considered in use and the packet is not sent. Thus if the CCA threshold is too low, no packet is sent because the channel is always considered blocked. If it is too high, the packet is sent, even when another radio is sending data, which increases the likelihood of collisions. In accordance with IEEE 802.15.4 the radio chip will also consider the channel in use when receiving "valid" IEEE 802.15.4 data, even if its signal strength is lower than the CCA threshold.

The ContikiMAC RDC protocol [Dun11] does not only evaluate the CCA when sending, but also when receiving: When sending a packet, ContikiMAC repeatedly sends the packet, until it gets an acknowledgment from the receiver. To receive packets, the radio is turned on in regular intervals. Once the radio is initialized, the radio is queried whether the channel is clear. This decision is based on the CCA decision of the radio chip. If the channel is clear, the radio is turned off again, as there is nothing to be received. If the channel is not clear, it is assumed that another node might be trying to send data and the radio is kept on to receive the next repetition of the packet. As a consequence, setting the CCA threshold too high does not only increase the chance of collisions, but also the chance of not detecting another node trying to transmit data.

This campaign targets how changing the CCA influences the RDC/MAC layer. Contiki however does not only support ContikiMAC, but also CX-MAC, an adjusted version of the X-MAC RDC protocol [Bue+06]. The experiment compares these two RDC protocols and how the CCA threshold impacts them in simulation and on the real testbed. The third supported option is SICSLoWMAC, which puts packets into 802.15.4 frames and does no RDC, but only simple CSMA (Carrier Sense Multiple Access) by default.

6.5.1 Experiment Setup

This campaign analyzes the impact of a hardware parameter, the CCA threshold, which is configured on the radio hardware during the initialization phase, and alternative software implementations of the RDC/MAC protocol. The experiment parameters were as follows:

- **Parameter 1:** CCA threshold: −55 to −25 in steps of 2; Default: −32
- **Parameter 2:** RDC protocol: ContikiMAC, CX-MAC and SICSLoWMAC; Default: ContikiMAC
- **Duration:** 20 min
- **Repetitions (Simulation/Testbed):** Erlangen: 50/10; Brunswick: 50/5

The configuration resulted in 2400 simulations for each testbed. Repeating the experiment 10 times in Erlangen resulted in 480 experiments, which took over a week to conduct. The 5 repetitions and 240 experiments, respectively, in Brunswick, still took almost 4 days to complete.

6.5.2 Results

Figure 6.10 shows the percentage of packets received at the sink, broken down by RDC protocol CCA threshold, simulation and testbed. The results from the simulation are shown as cross and are placed to the right, while the results from the testbed are represented as square and placed to the left of the corresponding discrete value. The medians of the experiments are connected with a line.

The results show that although there are deviations, there is a clear similarity between simulation and testbed. In the following I will discuss different aspects and their potential cause. It should also be noted that the numbers reflect the end-to-end communication. Thus, packets sent by nodes further away from the sink are less likely to arrive. Also, if an intermediate node looses the connection to the next hop, all nodes that send data through that node lose their connection to the sink.

CSMA All graphs show the effects of the CSMA protocol. While no packet is transmitted at CCA threshold of −55, the number of packets reaching the sink increases with higher CCA thresholds. While in Erlangen the fist packets are transmitted at −51 for simulation and tested, the simulations for Brunswick already transmit at −53, while the experiments require a threshold of −51.

This might have been caused by the time frame covered by the trace. The reproducibility of the effect however suggests a systematic error. A possible explanation is a combination of two effects: The radio chip has a hysteresis, which requires the signal strength to be even lower than the configured value before

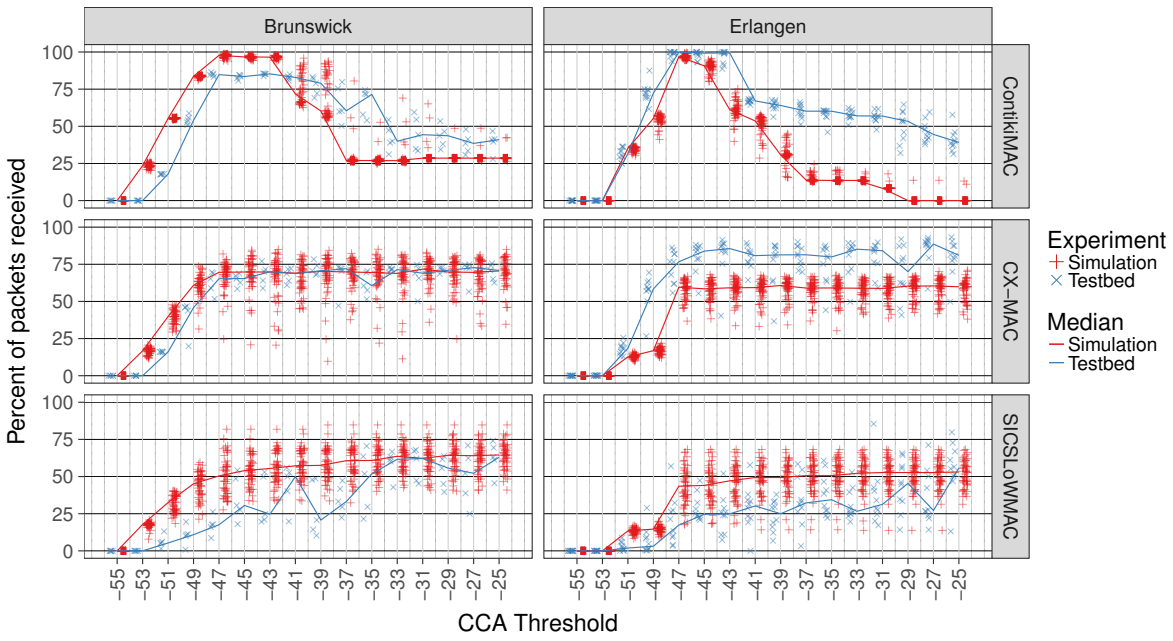


Figure 6.10: Percent of packets received at the sink, broken down by RDC/MAC protocol, CCA threshold and testbed. Each point represents the aggregated numbers of all nodes for a single experiment. To better distinguish simulation and testbed experiment they are arranged to the left and right of the associated discrete value. The line connects the median of all runs with the same configuration.

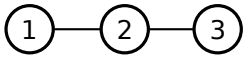


Figure 6.11: Hidden terminal problem: Node 3 cannot detect that node 1 is already sending data to node 2 and therefore causes a collision.

the channel is considered free. This is not implemented in the simulation[5]. Thus, the simulation is able to transmit data, even if the signal strength does not drop below the value required by the hysteresis. Additionally the testbed in Erlangen had a much higher density of WLAN access points and generally provided a noisier environment. This might have increased the averaged value sufficiently to compensate for the hysteresis. That the default hysteresis has an offset of 2, which is also the offset that can be seen for ContikiMAC and CX-MAC is an indicator for this theory.

ContikiMAC's drop at higher values, which is not related to a CSMA protocol, was already addressed in the introduction of the campaign and will be discussed separately. While it seems reasonable not to increase the CCA threshold too far to avoid collisions, at least for the presented scenario, actually increasing the threshold has little effect. In fact for SICSLoWMAC things even improve with higher values.

Most likely this improvement is caused by a combination of the channel not being utilized strong enough to see an effect of the hidden terminal problem. The hidden terminal problem is shown in fig. 6.11, where node 3 cannot detect that node 1 is already sending data to node 2 and therefore causes a collision and vice versa. In this case no CSMA protocol can avoid collisions.

While there is no visible improvement for the CX-MAC protocol after reaching a certain value – probably by effectively disabling the CSMA protocol, SICSLoWMAC does show continuous improvements. These can be explained with a flaw in the SICSLoWMAC implementation, which does not resend a packet dropped by the radio hardware based on the CSMA protocol.

ContikiMAC As already explained in the introduction of the experiment, ContikiMAC does not only use the CCA to decide whether to send a packet, but also whether to keep the radio on to receive a packet. Thus, if the CCA threshold is too high, it is not able to receive a packet. The effect of a higher packet loss at higher values is well visible in the graphs. However, the drop is not as strong in the testbeds, especially in Erlangen. This can be explained with the insufficient modeling of the radio (see section 3.6.9, page 61).

[5]Implementing this hysteresis is not reasonable without also implementing a realistic noise model as discussed in section 3.7, page 63.

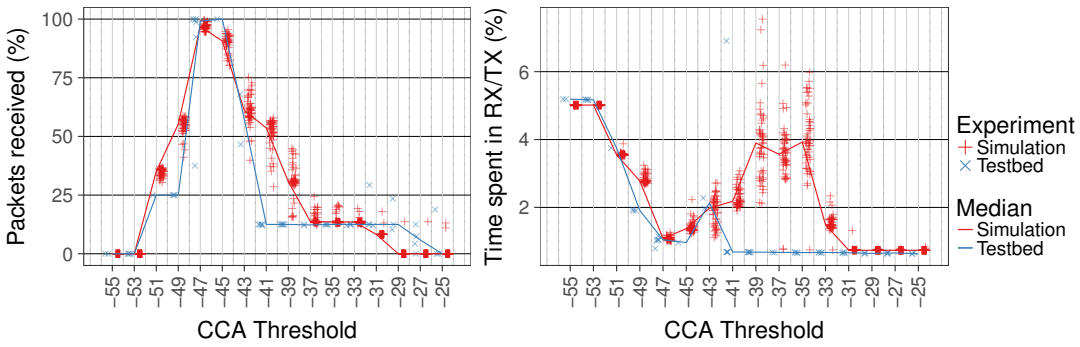


Figure 6.12: The experiment shown in section 6.5.2, repeated in Erlangen for ContikiMAC, yet with the CCA mode of the radio chip configured to only evaluate the signal strength.

The real radio considers the channel in use when receiving valid IEEE 802.15.4 data. For the nodes in the testbed it is sufficient to detect radio communication that is below the CCA threshold, but looks valid.

That it is not even possible for the simulated nodes neighboring the sink to send their data roots in the routing protocol. The protocol requires that routing information from the sink node is received, before the actual data is sent. Due to the issue with the CCA, this information is never received.

Detection of Valid IEEE 802.15.4 Data To verify the theory that the testbed performs better due to the CCA also taking "valid" IEEE 802.15.4 data into account, the campaign was repeated for ContikiMAC, yet configuring the hardware to only consider the signal strength. To reduce the run time the duration of the experiments was reduced to 10 min. Although this second experiment was made over a year later and there were slight changes in the positioning of the nodes, the similarity to the simulation becomes clearly visible in fig. 6.12. As in the simulation, the number of received packets first drops down to around 12.5 % and then to 0 at an CCA threshold of −25. Besides slight variations that can be accounted to the changes in the network, this is in accordance with the simulation. The same applies for the radio on time, which will discussed in the following. The peak that can be seen in fig. 6.13 at an CCA threshold of about −37 in the simulation is also visible in the testbed experiment, yet at −43 and much weaker.

CX-MAC For CX-MAC the numbers match very well in Brunswick, while in Erlangen the testbed performs better than the simulation. I was not able to find the cause for the offset. Most likely this is again caused by the insufficient radio model or the trace not perfectly resembling the average environment. The fact that the results in Brunswick match, and those in Erlangen do not, suggests that this is again related to the signal strength. The effect causing the drop for ContikiMAC is rather unlikely to affect CX-MAX: Instead of evaluating the CCA, the radio is kept on to receive a full packet. Dezfouli et al. suggest that the first bytes of a preamble are not required to successfully receive a packet [Dez+14b]. This might be a factor, yet it is questionable whether it is the only cause for the difference between simulation and testbed. Researching this would require an in-depth analysis of the implementation and goes beyond this work.

SICSLoWMAC SICSLoWMAC is not an RDC protocol, but solely a very simple MAC protocol without real CSMA implementation. Specifically it does not handle the situation where the packet is dropped due to the CSMA implementation of the hardware. This is most likely also the main cause for the reduced packet loss with an increased CCA value, as fewer packets are dropped by the hardware, but are still received despite the higher noise.

Energy Figure 6.13 shows the percentage of time spent with the radio turned on. This correlates to the energy usage of the radio, as sending and receiving has about the same energy consumption in the default settings of the radio chip used. SICSLoWMAC obviously has the radio turned on 100 % of the time. When looking very closely at the data for SICSLoWMAC one can see a very slight jitter and the median being above 100 %, which is caused by inaccurate accounting of Energest and can only be seen at all due to the high resolution of the graph.[6] The inaccuracy is however small enough to be neglected.

[6]The percentage of time with the radio on is calculated by summing up the time the radio is receiving or sending divided by the sum of the time the Central Processing Unit (CPU) spends in low power mode or is active. However, the transition is not atomic. Thus, when switching from receiving to sending the accounting for receiving is turned off and for transmitting turned on afterwards. If the timer increases in between those two operations the accounting becomes inaccurate. As the CPU switches into sleep mode more often than packets are sent, more ticks get lost, resulting in a radio-on time of over 100 %. This has been addressed in current versions of Energest (`8a7e2e58` [Con]).

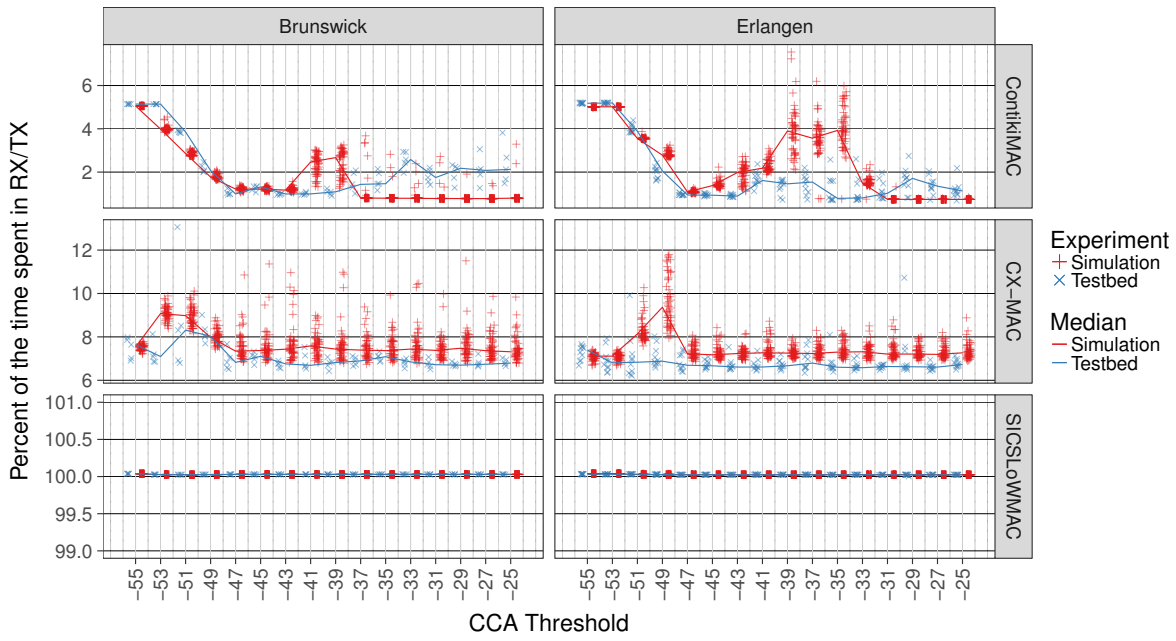


Figure 6.13: Percent of time spent with the radio turned on. As the energy consumption differs only slightly whether receiving or sending the states are not broken down. Each point represents the aggregated numbers of all nodes for a single experiment. The line connects the median of all runs with the same configuration.

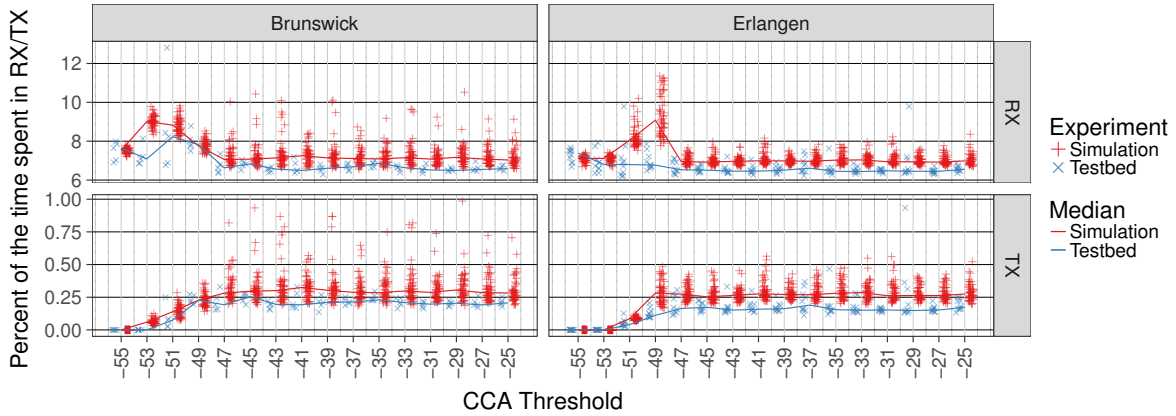


Figure 6.14: CX-MAC: Percent of time spent in RX or TX. Each point represents the aggregated numbers of all nodes for a single experiment. The line connects the median of all runs with the same configuration.

With an increasing CCA threshold, ContikiMAC's radio on time drops. This is caused by the radio not being kept on when no other node is sending. With exception of the offset in Brunswick and up to about an CCA threshold of about −41, simulation and testbed closely match. After that, the discussed inaccuracy of the radio-chip model starts showing its effects on the numbers generated by simulation. First by increasing the radio time, because transmissions become less reliable, then by keeping the radio off most of the time, because the connection to the sink is lost. As some nodes still have a connection to the sink in Brunswick, the effect is not as strong as in Erlangen.

Opposed to ContikiMAC, the simulated CX-MAC first increases the radio on time with a higher CCA threshold. This effect can also be seen in Brunswick's testbed, again with an offset of 2. In Erlangen however this effect is even more prominent in the simulation and not visible in the testbed.

Figure 6.14 shows the data for CX-MAC broken down to RX and TX. One can see that CX-MAC spends much more time listening than sending. The RX time starts at a certain level, then spikes up and drops to a lower stable level. At low CCA thresholds the CSMA algorithm puts the radio in RX mode to check whether the channel is free. This value is higher than in normal operation, because the higher network layers try to unsuccessfully send their packets again and again. The TX time increases until the

CCA-threshold is above the background noise for all nodes, and they can therefore successfully send their packets. Consequently the spike in the radio time can clearly be accounted to the RX time.

To understand the spike in the RX time, a short introduction to the behavior of the network stack is required. CX-MAC works similarly to ContikiMAC: The radio wakes up regularly to listen for potential packets. To send a packet to a node, the routing protocol first sends broadcast packets to discover a route. In broadcast mode, CX-MAC sends the discovery packet repeatedly until all neighbors should have received the packet.

The answer to that request is sent using unicast packets to the querying node. In this case the packet is repeatedly sent until an acknowledgement is received or the send operation times out.

Once the CCA threshold is high enough, the CSMA mechanism will not keep selected nodes from sending broadcast packets. The other nodes can receive these requests as the CCA threshold does not impact the reception for CX-MAC. The nodes will then try to send their answer, which will be dropped because the CCA threshold is still too low for them. Contiki's implementation of CX-MAC does however not evaluate whether the packet was successfully sent and waits for an acknowledgement. This causes these nodes to spend additional time checking for the channel being free to send packets and also to wait for an acknowledgement. Once the noise level drops below the CCA threshold, the RX time drops for those nodes, too.

In the real world the effect is not that bad, because the background noise changes and is likely to drop below the CCA threshold again. Especially in Erlangen this is likely to happen due to the noisy environment. Yet in the simulation, the background noise does not change for a sampling epoch, which is 80 sec for the presented experiments. Thus, it can be that two nodes can communicate well for 80 sec and for the next 80 sec one node cannot send any data anymore, because the background noise has risen above the CCA threshold. This aggravates the problem of establishing a route and not being able to send data using it and thus causes the spike.

6.5.3 Discussion

The results clearly show that the overall approach of using simulation to optimize WSN software does work. The default CCA threshold of the hardware was -32 and ContikiOS provided no generic way to adjust it[7]. Based on the numbers presented, this default value yields inferior results, both in simulation and testbed. Nonetheless there are plenty of publications using ContikiMAC on the Sky-mote hardware, often investigating higher level protocols that do not mention changing this value. Thus they either tuned this value without mentioning it, had significantly different network characteristics, or the results presented are not as good as they could have been.

Due to the issue with the inaccurate CCA mode model of the radio chip, the overall results are not as accurate as hoped for. However degrading the real world to improve the accuracy of the simulation does not seem reasonable, and as discussed, modeling the behavior requires more research. Despite these issues the qualitative characteristics of the protocols when changing the CCA value were captured.

Based on the similarity of the results from the two testbeds and the deviation between simulation and testbed, it could be argued that the advantage of running network-specific simulations is minimal. Yet this campaign mainly looked at the CCA threshold, and, as discussed, there are major issues, especially with the noise model. Also, the two testbeds were very similar. Both are located in an office environment and distributed over multiple rooms, opposed to other testbeds that are placed in a single room or outside. Despite all this, for ContikiMAC the simulations of the two locations yielded distinct results that clearly correlate to the corresponding testbed: The plateau of good connectivity is much wider in Brunswick than in Erlangen.

6.6 Campaign: Packet Rate Versus Channel Check Rate

An issue that motivated this work is the interaction of configuration options between layers. In the previous campaign the best results were achieved using the ConikiMAC protocol. As a CCA threshold of -45 performed well in both testbeds and the simulation, this was used for this campaign. To show the inter-layer interaction, this campaign compares the impact of the packet rate, which is configured at the application layer, and the CCR at which ContikiMAC wakes up to listen for potential packets. The assumption is that when more packets are sent, the network eventually goes into an overload situation.

[7] As part of the patch submitted upstream to make this configurable at compile time (`c67c048c` [Con]), the default value was set to -45.

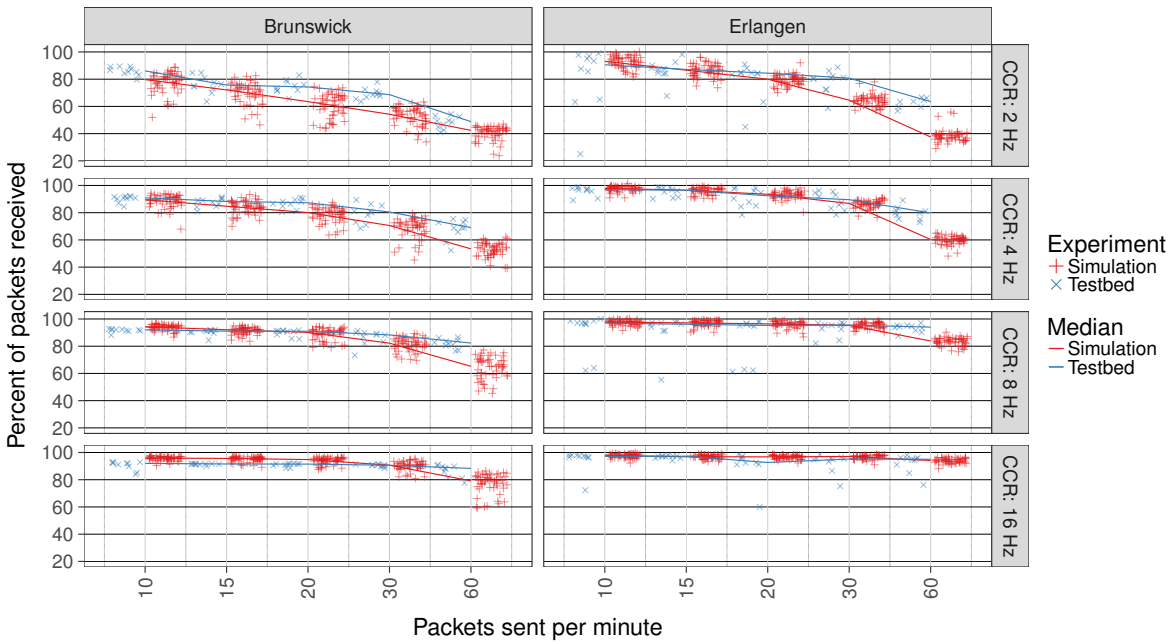


Figure 6.15: Packet Reception Ratio (PRR) at the sink based on the amount of packets sent by the clients and the Channel Check Rate (CCR) of the ContikiMAC protocol. Each point represents the aggregated numbers of all nodes for a single experiment. To better distinguish simulation and testbed experiment, they are arranged to the left and right of the associated discrete value. The line connects the median of all runs with the same configuration.

However, if the channel is checked for potential packets more often, the available bandwidth can be increased.

Additionally to changing of the configuration options, the duration of the experiment was reduced to 5 min to reduce the overall duration of the campaign.

6.6.1 Experiment Setup

This campaign analyzes the impact of the packet rate, which is an application layer configuration, and the CCR which is a configuration option of the RDC/MAC layer. The experiment parameters were as follows:

- **Fixed:** RDC protocol: ContikiMAC
- **Fixed:** CCA threshold: −45
- **Parameter 1:** Packet Rate: 10, 15, 20, 30 and 60 1/min.
- **Parameter 2:** CCR: 2, 4, 8 and 16 Hz; Default: 8 Hz
- **Duration:** 5 min
- **Repetitions (Simulation/Testbed):** Erlangen: 50/10; Brunswick: 50/10

The configuration resulted in 1000 simulations for each testbed. Repeating the experiment 10 times resulted in 200 experiments taking ≈22 h.

6.6.2 Results

Figure 6.15 shows the percentage of the packets sent by the clients that arrived at the sink. They are broken down by CCR, packet rate, location, and whether the results are generated by the testbed or the simulation. The visualization is the same as in the previous graphs: The results of the experiments are placed to the left and right of the discrete values. A line is used to connect the median of each combination of configuration options.

The overall results form the experiment are as to be expected: Increasing the number of packets sent brings the network into an overload situation and eventually increases the percentage of packets lost. As

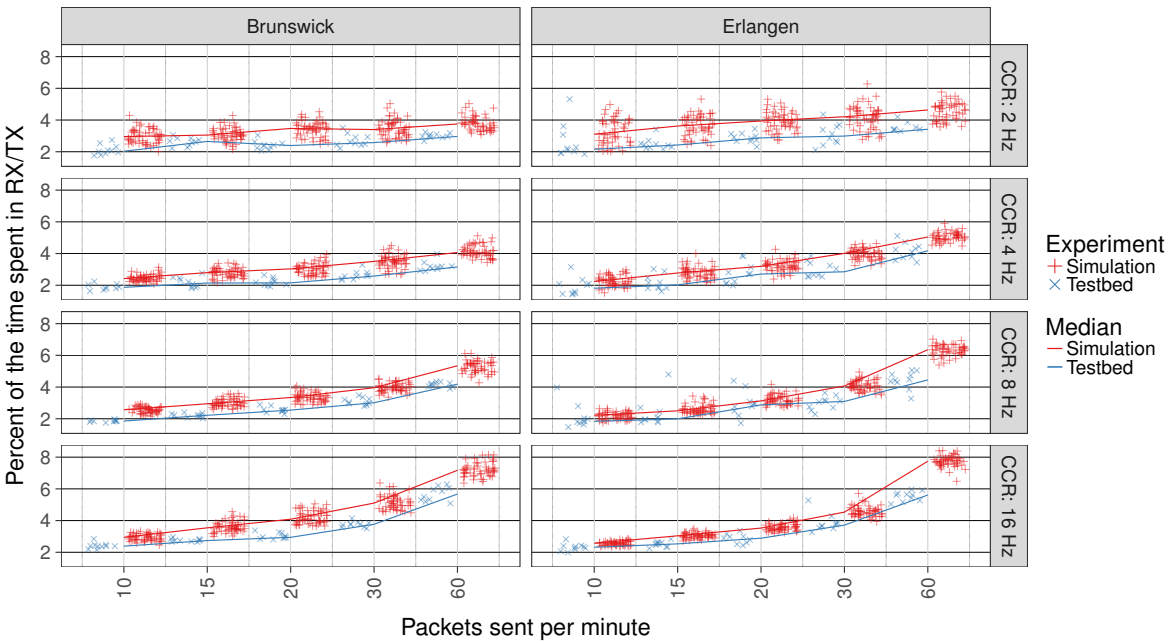


Figure 6.16: Radio on time based on the amount of packets sent by the clients and the Channel Check Rate (CCR) of the ContikiMAC protocol. Each point represents the aggregated numbers of all nodes for a single experiment. To better distinguish simulation and testbed experiment they are arranged to the left and right of the associated discrete value. The line connects the median of all runs with the same configuration.

the network in Brunswick is larger, this happens earlier than in Erlangen. Also, increasing the CCR does increase the number of packets that successfully reach the sink.

As in the experiments investigating the effects of changing the CCA threshold, the testbeds perform better than the simulation. This becomes visible at an CCR of 2 Hz, where the overload situation is especially strong. It is hard to say how this correlates to the previous experiment, where the results of testbed and simulation were very similar when using ContikiMAC at a CCA threshold of −45.

Figure 6.16 shows the time spent with the radio turned on, which as discussed, relates to the energy usage. At a low packet rate of 10/min, a CCR of 8 Hz, the default, yields the best results. It can only be speculated about the reasons, why the numbers do not further improve when reducing the CCR to 16 Hz. As ContikiMAC is a rather complex protocol with many options that can be tuned, it is not unlikely that other parameters are optimized for the default CCR value and need to be adjusted to gain better results at other CCR values.

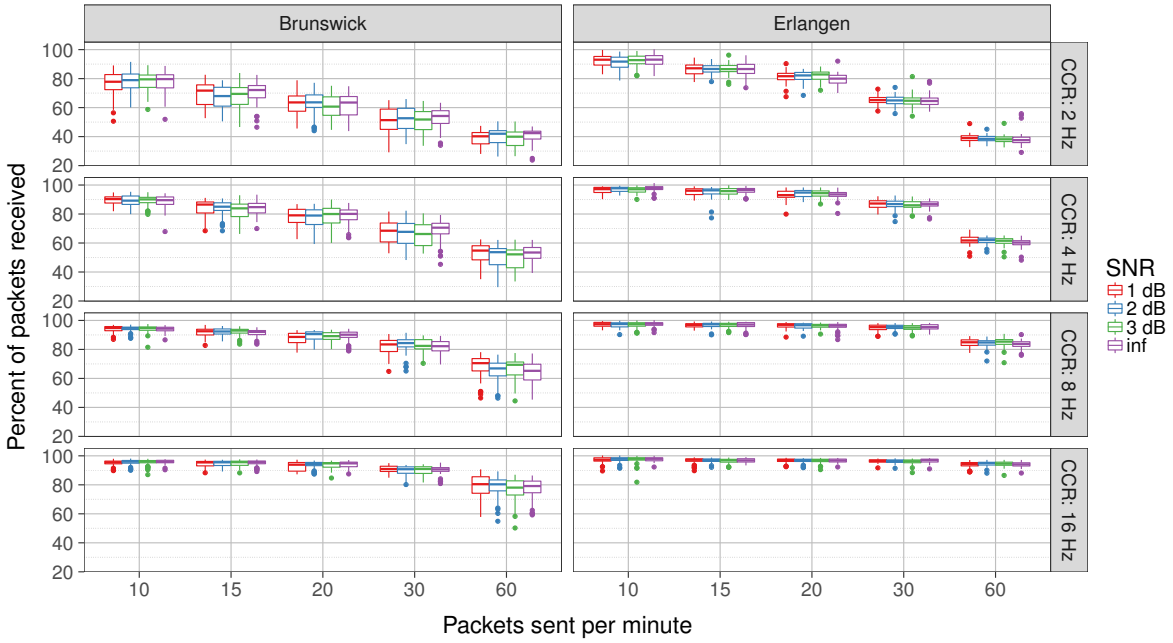
With the number of packets sent, the radio on time increases, too. At a packet rate of 60/min the best results can be achieved using a CCR of 2 Hz, both in terms of packet loss and energy consumption. This is likely to be caused by the reduced number of retransmissions.

Again the results of the two testbeds are very similar, however at a CCR of 2 and 4 Hz Erlangen spends more time with the radio turned on in the simulation and testbed. At higher CCRs this is not as visible anymore.

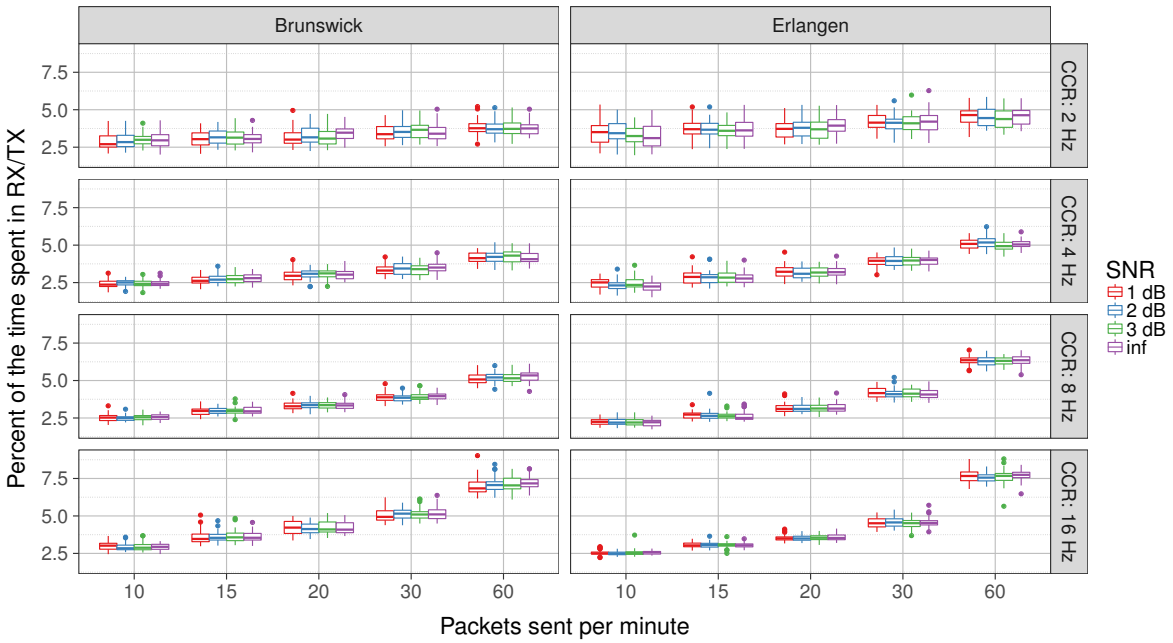
6.6.3 Capture Effect

Especially at high rates collisions caused by the hidden terminal problem (see fig. 6.11) are more likely to happen. To evaluate whether the lack of support for the capture effect causes the deviation between simulation and testbed, a prototypical support was implemented for Cooja/MSPSim. Simulations were run with no support for the capture effect, and a required SNR of 1, 2 and 3 dB to successfully receive a packet. The results are presented in fig. 6.17. Not supporting the capture effect is the same as requiring an infinitive SNR and is therefore labeled as “inf”.

For the high packet rate of 60/min is seems like slightly better results can be achieved for the PRR (fig. 6.17a). Yet the difference is so small that this has no significance. Especially, as under certain combinations of packet rate and CCR support for the capture seems to worsen the results (e.g. Brunswick, CCR: 2 and 4 Hz, packet rate: 30/min). The same can be seen for the time in which radio is active. The difference between simulations with and without capture effect are so small that they have no significance.



a) Packet Reception Ratio



b) Energy

Figure 6.17: The plots show simulation results from the same experiment as presented in fig. 6.16, yet comparing the default radio model with prototypical support for the capture effect. The SNR is the distance of the signal to other sources of noise to successfully receive a signal. The default model does not support this at all, which equals a required SNR of ∞. The hinges of the box plot extend to the 25th and 75th percentiles. The whiskers extend to the highest value within 1.5 times the inter-quartile range. All other values are printed as outliers.

In conclusion, for ContikiMAC the capture effect is insignificant – at least in this setup. It therefore cannot explain the difference between simulation and testbed.

6.6.4 Discussion

Although the results do not yield groundbreaking insights, they do support the approach. The results of simulation and testbed do differ, yet there is a clear similarity. Also the two locations of the testbeds have the same effect on testbed and simulation: Reaching the overload situation earlier in Brunswick and using more radio on time at low CCRs in Erlangen.

Also the effect of the lowest energy consumption for low data rates at a CCR of 8 Hz is interesting. However trying to find a second parameter that allows to reduce the energy consumption at lower CCRs requires DryRun to be extended with some heuristic.

6.7 Campaign: Software Versions

WSN software cannot only be tuned in terms of configuration parameters, but also different software revisions can yield different results. To test how Contiki developed over a period of 1.5 years, 41 revisions were selected.

Contiki is developed using the GitHub flow model [Git16]. The basic idea is that new features are developed on a separate branch, and only working versions are merged into the master branch. Thus, while feature-branches can contain versions that do not work, the master branch should always contain working versions. Due to this work flow, all commits from the master branch were taken and split up into 40 sections of approximately the same number of commits.

This experiment was also simulated twice using two different traces, which were already introduced in section 6.3.1, page 101. Both traces were taken right after the experiment, one with the default resolution of sending 8 packets/episode and the other using a higher resolution of 32 packets/episode. By chance the connectivity of the trace taken with the higher resolution had a longer phase with very bad connectivity. This also impacts the simulation results, as the high-resolution trace does not show a better resemblance of the testbed results, as one might expect, but a worse.

6.7.1 Experiment Setup

This campaign analyzes how Contiki evolves over the duration of 1.5 years. To make the different versions comparable two already discussed patches ware applied if necessary.

- Correct initialization of Energest: `c0783e28` [Con]
- Allow using the serial in an 6LoWPAN environment.

The experiment parameters were as follows:

- **Fixed:** RDC protocol: ContikiMAC
- **Fixed:** CCA threshold: −45
- **Fixed:** Packet Rate: 3/min.
- **Fixed:** CCR: 8 Hz
- **Parameter 1:** Contiki Version
- **Parameter 2:** Trace (normal and high resolution)
- **Duration:** 13 min
- **Repetitions (Simulation/Testbed):** Erlangen: 50/10

6.7.2 Results

Figure 6.18 shows the PRR of the testbed and the simulation using the trace with the high and the low resolution. The data is visualized using a box plot that ranges from the first to the third quartile. It's whiskers range to the most extreme value that is within 1.5 times the box size. All values beyond 1.5 times the box size are marked as outliers.

That the trace with the high resolution had a phase of very bad connectivity resulted in the 1st quartile often being below a PRR of 50 %. Generally the connectivity during this experiment changed much

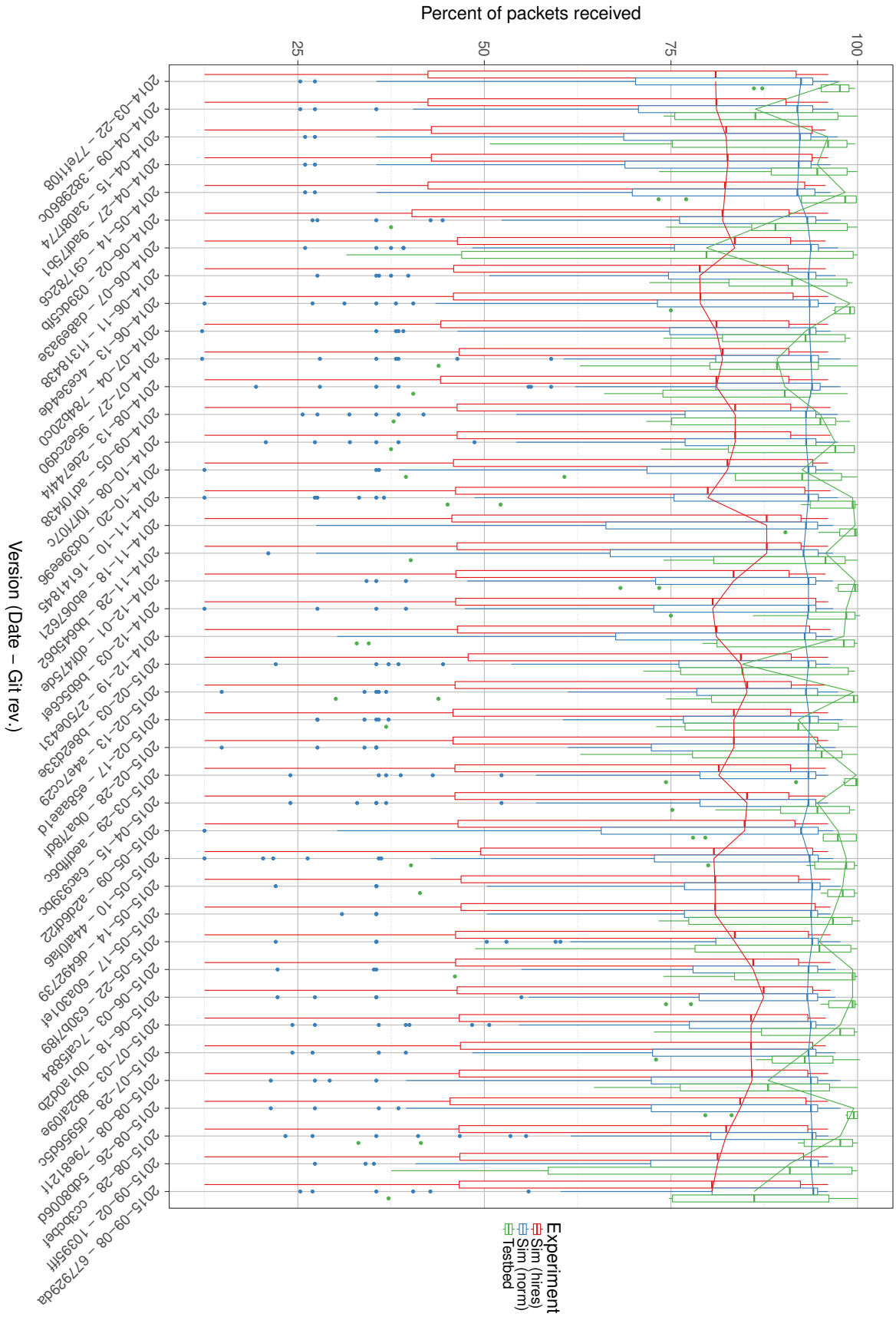


Figure 6.18: The PRR of packets sent by the clients that arrive at the sink based on Contiki's version. The data was collected from the testbed, a simulation with a "high" resolution trace, and a simulation using the "normal" resolution, as was also used for the other experiments. The box ranges from the first to the third quartile. The whiskers range to the most extreme value that is within 1.5 times the box size. Values beyond that are marked as outliers (dots). The median is connected with a line.

stronger, than when the other experiments were performed. This resulted in the results spreading much stronger.

Due to the strong variations in the PRR, it is not possible to draw any real conclusions from this data. Even though, comparing the median of the normal simulation with the median of the testbed there seems to be some correlation. This is however most often off by one. When using some linear configuration option, it can be argued that this is caused by some effect not accurately modeled and therefore being off by one. In this case however, there is no correlation between the software revision and the resulting firmware that would allow arguing that the data is off by one. The effects seen must therefore be considered random.

Figure 6.19 shows the radio on time of the same experiment. Up to the release at 2014-10-08 the numbers are relatively stable with the testbed performing best, followed by the high resolution trace and the normal resolution trace. That the energy usage for the trace with the normal resolution is higher, is most likely due to more packets having to be forwarded using a connection with an average packet loss. This will be looked at in more detail later.

Starting with the revision from 2014-10-20 the numbers degrade significantly, with the median radio on time of the testbed being about twice as high. After this change the median does vary more than before that change, but stays relatively stable. Additionally, the distance of the quartiles increases, resulting in larger boxes in the figure. The change is strong enough that it cannot be accounted to random effects. In the following I will refer to this change as "energy-regression".

Both, the median increasing and the quartiles moving apart after the energy-regression can also be seen in the simulation. Interestingly, the high resolution trace starts outperforming the testbed after the energy-regression. This hints at the regression correlating to the number of packets being transmitted. Thus, the radio-on time required to transmit a packet has increased. Finding the exact cause is however out of the scope of this work, which does show that is possible to find such a change using simulation.

Figure 6.20 and fig. 6.21 show the same data, yet broken down by node. They show some additional effects.

Packet loss I will first look at the percent of packages that arrived at the sink (fig. 6.20). For node 8, which has good connectivity, testbed and simulation results match almost perfectly. This does not apply to the other nodes, for which the results yielded by the simulation vary strongly, due to the nature of the traces taken. In comparison to the summarized results of fig. 6.18, the median of the individual nodes look a bit more stable. Although some of this must be accounted to the different scale of the graphs, this is mainly due to not aggregating the following three groups of nodes:

- Node 8, which has an extraordinary good connection.
- Nodes 3 and 4, which are not connected to any additional node, but the sink.
- The rest of the nodes, which need to route through node 8.

While the graph yields little additional information in terms of comparing testbed and simulation, I want to explain some artifacts that can be seen in the graph. Just looking at the visualization of the graph two things stand out. Firstly it seems like node 8 has no box plots. Due to the good connectivity, node 8 had very little variations, so that these can only be seen when zooming into the graph. Secondly, the results generated using the hi-resolution trace vary strongly. This is caused by the strong environmental changes while taking the trace, which have been discussed before.

When breaking the data down by node, the network topology (fig. 6.22) becomes visible. As the experiments were taken at a later point in time, with some nodes being relocated within the room, the topology slightly differs from the one during the other experiments (fig. 6.8, page 108). The main difference is that node 6 has no connection to node 8 anymore, despite being placed only about 20 cm apart from node 9. In consequence all data of nodes 5, 6, 7 and 10 must be routed through nodes 9 and 8. In the graphs this is especially visible in fig. 6.20 for the revision 2014-06-07 and 2015-09-08. While the connectivity for almost all other revisions to all these nodes is very good, those two have some testbed experiments with a high packet loss. As the other box plots look similar, the most likely cause for this effect is that the connectivity between nodes 8 and 9 was degraded during those experiments. In consequence all nodes that had to route through this connection were equally degraded.

For the high resolution trace there were phases during which nodes 8 and 9 had bad connectivity, too. This influenced just one quarter of the experiments. In consequence the first quartile of the PRR of the nodes connected through node 9 is below 30 %, while the mean is at around 80 %.

Radio on Time Breaking down the radio on time by node (fig. 6.21) does show some additional aspects in terms of comparing testbed and simulation. Just as in the summarized fig. 6.19, the energy-regression of revision 2014-10-20 is also visible.

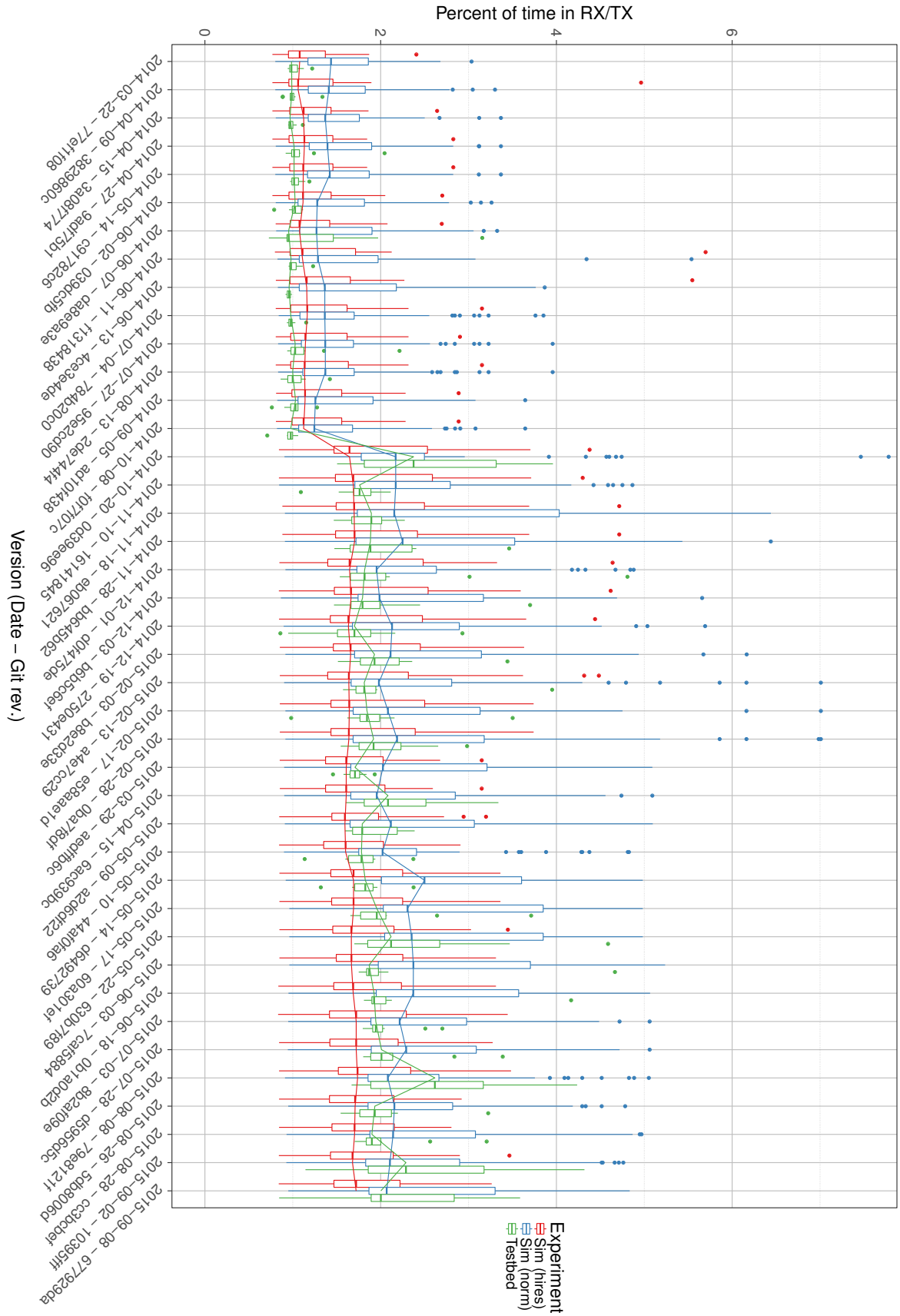


Figure 6.19: The radio on time based on Contiki's version. The data was collected from the testbed, a simulation with a "high" resolution trace, and a simulation using the "normal" resolution, as was also used for the other experiments. The whiskers range to the most extreme value that is within 1.5 times the box size. Values beyond that are marked as outliers (dots). The median is connected with a line.

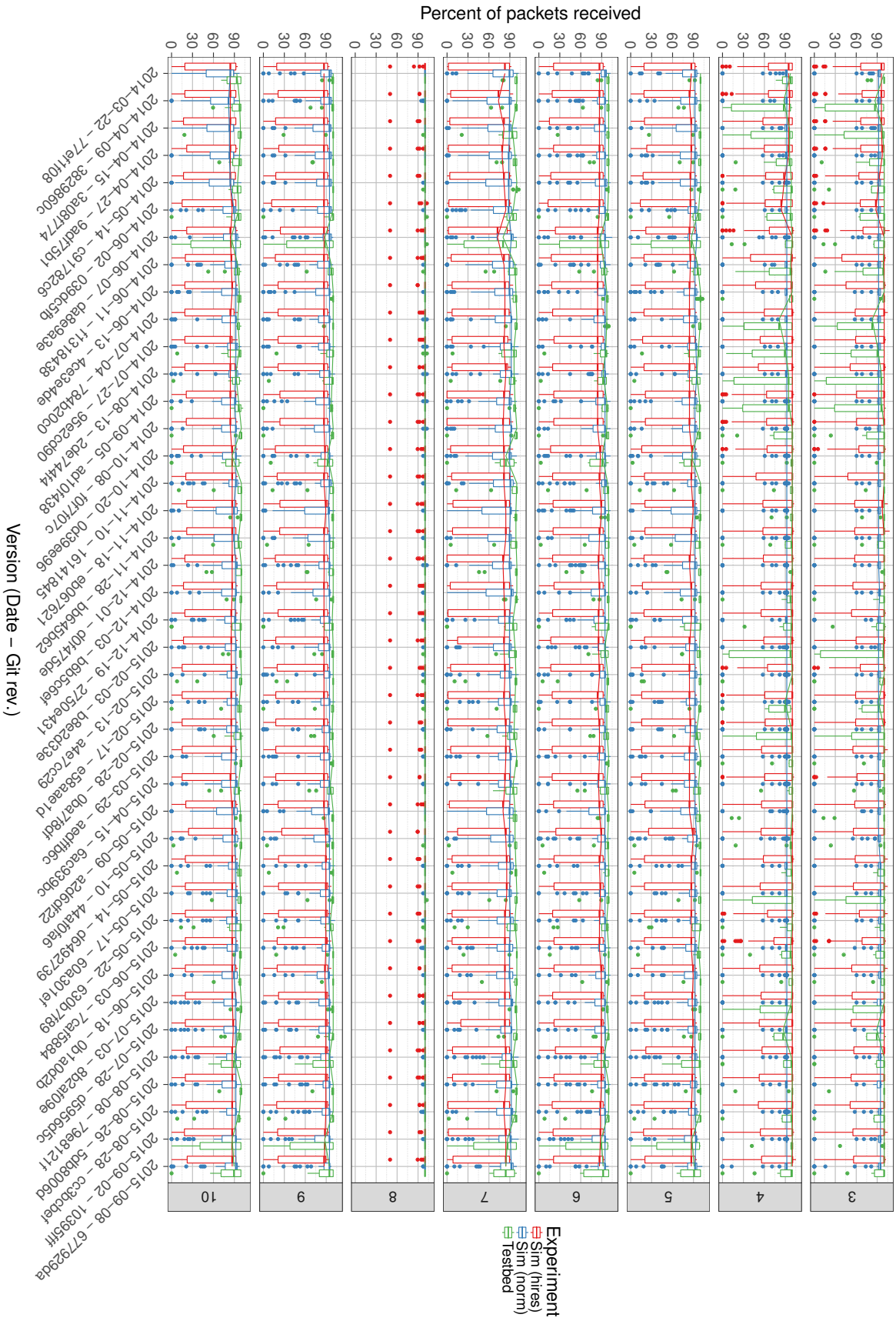


Figure 6.20: The same data as in fig. 6.18, yet broken down by node.

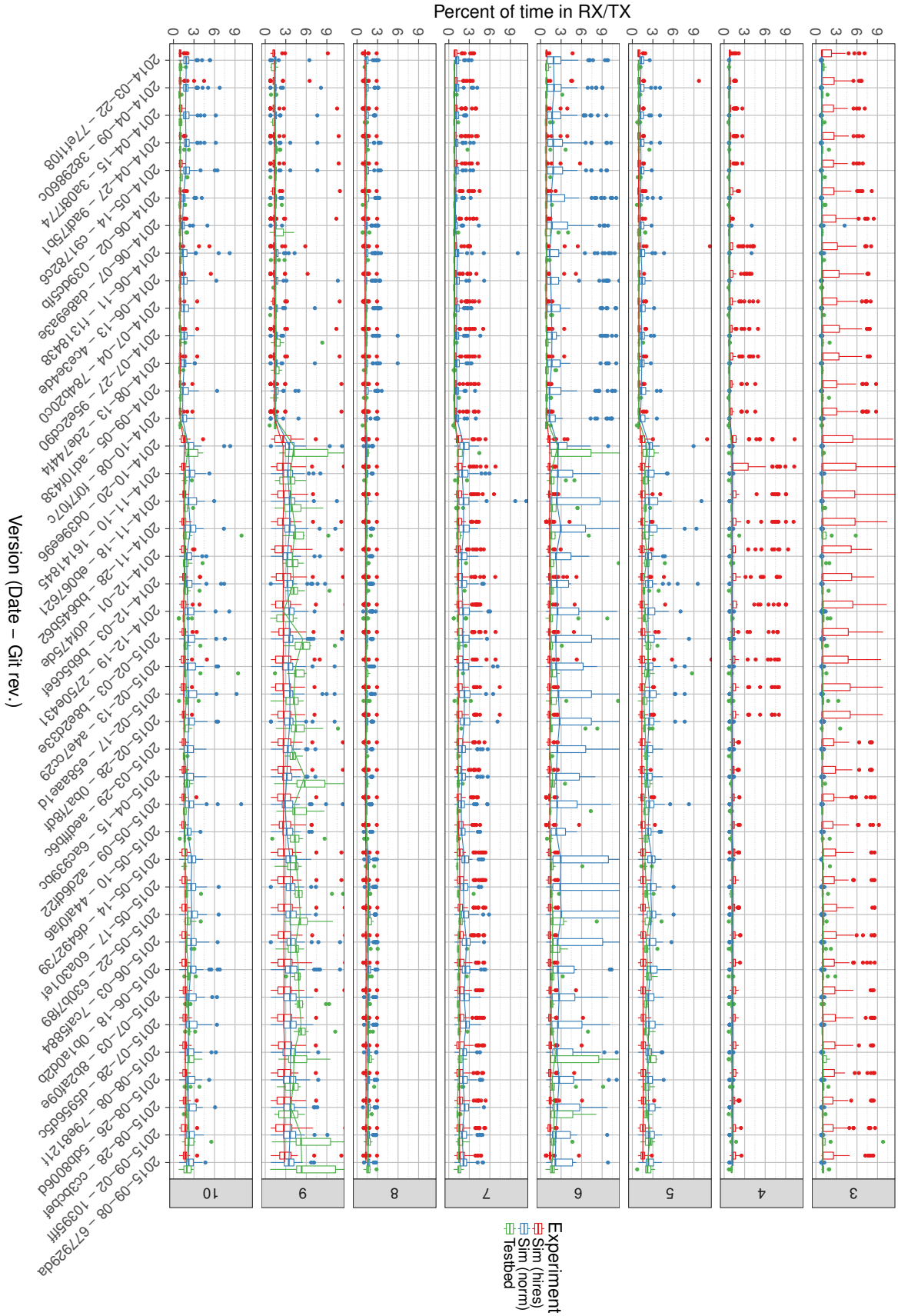


Figure 6.21: The same data as in fig. 6.19, yet broken down by node.

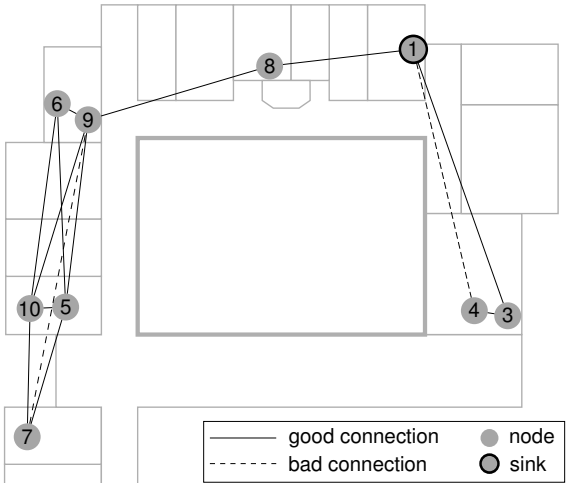


Figure 6.22: Layout of the WSN testbed in Erlangen during the experiments comparing software revisions. Note that compared to fig. 6.8, page 108 there is no connection between nodes 6 and 8, nor between 9 and 1 and the connection between nodes 1 and 4 is not good any more, but bad. Also nodes 9 and 7 now have a bad connection whereas they had no connection in fig. 6.8.

Before the regression the results are very stable for all nodes, for the testbed and the simulations. Exceptions are node 6, where the normal resolution trace yields inferior results, and node 3, where the 3$^{\text{rd}}$ quartile has a significant offset, while the median maps the testbed results. Both can be explained with the characteristics of the traces.

Looking at the results after the energy-regression, nodes 4 and 8 differ from the summarized results. Both are almost unaffected by the change, in none of the simulations nor the testbed. As they both have good connectivity to the sink, it can be assumed that only connections with inferior connectivity are affected by the regression. The same applies to node 3 for the testbed and the trace with the normal resolution.

Nodes 5, 7 and 10 are only slightly affected by the regression, yet to a similar degree for both traces and the testbed. To some extent this also applies to node 6, yet with exception of the trace with the normal resolution. The normal resolution trace, which is already affected before the energy-regression, is much stronger affected by the energy-regression than would be expected based on the data before the regression. Unfortunately the cause in unclear, and would require further research.

Testbed and simulation certainly match best for node 9. Here the regression has a very similar effect on both simulation traces and the testbed. Also, not only the median shows the regression, but also the 1$^{\text{st}}$ and 3$^{\text{rd}}$ quartile move apart by a similar amount.

6.7.3 Discussion

This campaign differs to the previous campaigns in two aspects: Firstly it was taken about a year later. And secondly it did not test a configuration option using different values.

During that year the network attributes changed significantly. While the connectivity did change while executing the other campaigns, it did not change as strong as during this campaign. Especially that some connections range from very good to non existent, disconnecting part of the network from the sink, is problematic. The changes also cause the two traces to differ so strongly that it was not possible to use them for their original intent: Comparing the effect of the resolution of the trace.

While it would have been possible to take another set of traces that yielded better results, this would have not answered the question, whether higher resolution traces yield better results. The only way to achieve this is to sample a high resolution trace down to a low resolution trace, so that they reflect the same time frame. And even then the significance would have been limited:

Assume that the trace with the higher resolution yields overall better results. If the conditions during the trace were better than during the testbed experiments the low resolution trace would be closer to the testbed results, while if they were worse it would be the other way round. Thus, the significance would be limited. In this case however, by chance, the trace with the low resolution yields significantly better results and, again by chance, reflects the testbed better.

This highlights the importance of selecting a suitable trace for the network. Although it worked out quite well for the other experiments, using a trace of a week and generating synthetic traces reflecting the network characteristics might actually yield more realistic results than raw traces.

Another aspect, where this campaign differs from the others, is the type of input data. Iterating through configuration options allows detecting trends like: Towards a certain value things get better and after reaching another value it gets worse again. With software revisions this is not possible. Each version must be seen as independent. It is also possible that two revisions with many differences produce exactly the same results, because all the changes were made to code that is not used in the experiment.

Nonetheless, between 2014-10-08 and 2014-10-20 a change was made that affected all the following revisions, causing a significantly increased radio on time, especially for nodes with suboptimal connectivity. Interestingly, this does not affect the PRR. It can therefore not be explained with an increased packet loss or similar.

When comparing the results of the testbed and simulations on a per-node basis, the regression clearly shows a proportional correlation: If the simulation results were worse than the testbed before the regression, they were still worse afterwards. The same applies to the variations, thus the size of the boxes. If they were larger before the regression, they are still larger after the regression, even though the regression generally also increased the variation of the results.

This again shows the power of the approach using traces. If the network consisted of only good connections, the regression would not have mattered, neither in the testbed nor the simulation. Thus, although the simulation results do not perfectly reflect the testbed and some results differ significantly, the simulation is still able to show that a change was made in the code that impacts this specific network.

6.8 Conclusion

The evaluation shows that the simulation is accurate enough to optimize WSN networks. The simulation results are however not accurate enough to provide one with concrete values, but rather estimates. Simulations are therefore not suitable to fine-tune a WSN, but well suited to get a rough understanding of how a parameter affects a WSN and in which region an optimal parameter lies. Considering the time required to execute a multidimensional WSN optimization, this is a huge advantage.

The evaluation also showed that it is important to not only look at any network, but the specific network in question. This became especially visible in the campaign comparing different software versions (section 6.7, page 118), where the discussed regression only applies to nodes that have bad connectivity towards the next node. It therefore most likely does not affect networks that are well meshed.

That specific campaign also highlights the importance of selecting a suitable trace. In this experiment a high and a low resolution trace were used. The high resolution trace yielded inferior results. This was not due to the higher resolution, but was due to the trace not reflecting the typical state of the network while the campaign was run on the testbed.

During the evaluation yet another bug in the simulation model of the radio chip was uncovered that was not found in advance: The CCA of the radio chip is not correctly modeled (section 6.5.2, page 110). It is likely that there are more issues with the simulator, especially as multiple components interact with each other and documentation is sometimes vague. Consequently, while the presented approach provides significant support to optimizing a WSN, the simulated results must not be trusted and always be verified using real hardware and the concrete network.

7 Summary and Outlook

The motivation to analyze the feasibility of optimizing Wireless Sensor Networks (WSNs) using deployment-specific simulation was driven by two consecutive problems:

1. **Complexity of configuration:** An optimal configuration is crucial for the robustness and lifetime of a WSN deployment. Finding such an optimal configuration can easily become a challenge, as modern WSN Operating Systems (OSs) provide a huge amount of configuration options, especially at the network layer. Often a configuration option's impact and the interaction with other parameters can only be judged by an expert. For the average WSN application developer this often only leaves the trial and error approach. As there typically is only one testbed available, this can be very time consuming and the high amount of random effects in WSNs make it even more tedious. Besides, the testbed often does not resemble the target deployment.

   Simulation could be a solution to this problem. It is possible to run many simulations in parallel, allowing running a huge amount of experiments within a short time frame. The limitations are the maximum runtime of an experiment in the simulation and the available computation power. However the latter is readily available when using computing clusters or cloud-services. This however introduces the second problem:

2. **Realism of simulation:** While simulations are indispensable for testing and debugging WSN applications, the results must be treated with caution. Any numbers generated, must be verified using real hardware. These deviations of the simulation have three causes:

   - Bugs in the implementation of the simulator
   - Inaccurate models
   - Insufficient configuration

   The first two can only be addressed by the developer of the simulator. The last is in the responsibility of the user. However, most models are either over-simplistic, or have a huge configuration space. To set a suitable configuration requires a profound knowledge of the network attributes in the target environment. Using a configuration that was previously successfully used in a similar environment, for example another office or forest, can improve the realism. Yet even if the environment is similar, the topology is likely to differ. A closely meshed network is much more inclined to compensate strong changes in connectivity by rerouting data using other nodes. In the end there is often no getting around making a survey of the specific deployment. Without tool support this can easily become tedious work.

The analysis of these problems showed that while the hardware can mostly be simulated with high accuracy, the network is often oversimplified. This is rooted in the problem that even slightly more complex models often require a too complex configuration to be used reasonably. I addressed this by developing two sets of tools called RealSim and DryRun. RealSim traces a network and replays the trace in the simulator by configuring the model in accordance with the traced network. DryRun allows running experiment campaigns using a user-defined parameters space, both using simulation and the testbed.

Using DryRun to compare the effects of changed parameters on the testbed and the RealSim-enhanced simulation, did not only allow me to evaluate the feasibility of the overall approach, but also revealed multiple bugs in the simulator, the OS and network protocols used, as well as additional weaknesses in the radio-related models. Some of these were fixed, while others require further investigations that go beyond the context of this work.

Most importantly the evaluation shows that the approach is feasible. And even if the simulation is often not accurate enough to make experiments on the real network redundant, it can still reveal the characteristics of the configuration options and thus help immensely to reduce the potential configuration space.

Besides original intention of optimizing WSN software, there are two other use cases for DryRun and RealSim: By making the simulation more realistic, they can help understand why a specific WSN behaves the way it does, as simulators provide better and easier to use debugging capabilities than real hardware. Secondly, by running comparable experiments in the simulator and the testbed it is possible to reveal bugs in the simulator by analyzing the discrepancies between testbed and simulation.

7.1 Contributions

This work contains two main scientific contributions:

Analysis of realism in WSN simulation
As part of this work I did an analysis of the different aspects that influence realism in WSN simulation. Many other works only looked at one specific aspect and blind out other aspects. As a result models are based on unrealistic assumptions and evaluations come to wrong conclusions. To better asses the realism of simulation, I tried to give a holistic overview that concentrates on showing how a certain aspect influences the realism of simulation.

Improving the realism of WSN simulation by mapping real networks to the simulator
By developing RealSim I provided a tool that allows mapping traces of real networks to a simulator. While others tried this approach before, those only concentrated at Network Level Simulators (NLSs) and used very controlled environments as reference. I showed that these limitations make the presented solutions unsuitable for real world applications. By combing the approach with an Instruction Level Simulator (ILS) and evaluating it using an office environment, I show that my approach can be applied to real WSN applications in dynamic environments.

During the research I developed multiple tools, specifically DryRun and RealSim. All of these are accessible for other researchers to use and evaluate. [Dat; rea; Dry] This work also uncovered multiple bugs and limitations of the used simulator. A far as possible these were fixed and sent to the developers of the simulator.

7.2 Limitations and Future Work

In my work I tried to cover as many functions as possible by using rather complex network protocols, 6LoWPAN (Internet Protocol version 6 over Low power Wireless Personal Area Network) and ContikiMAC. As the evaluation shows, the results yielded by simulation and testbed show some discrepancies. These could be traced down to certain inaccuracies in the simulation model. However, the experiments were only executed on a single node-type, and although I ran my experiments on two different testbeds, both were located in an office environment. It seems reasonable that radios that provide similar information will also yield similar results. Also offices already provide a somewhat hostile environment, and it is likely that it is possible to gain more accurate results for testbeds placed in a more quiet environment like a forest. Until this is verified using real hardware, there is no guarantee that this is the case. It is possible that such an environment triggers effects that were not detected because they were not used, or because their impact was mitigated or covered by other effects.

In the following I will look into the limitations and discuss how these can be addressed in future work.

7.2.1 Conceptual Limitations

Although currently the limitations of the simulator and lack of appropriate models are the main impediment of the approach, there are also some conceptional issues. The RealSim enhanced simulation relies on a

trace of the target network. Thus the simulations only reflect how the network might have behaved during the time-frame when the trace was taken. If the trace does not cover a typical situation, the value of the simulation results is limited. Even more so, if the environment changes, the trace becomes worthless and actions based on the simulation results might degrade instead of improve the situation.

Future Work: Continuous Monitoring Any WSN sends some data over the network. This data that is sent anyway can be used for a low-impact monitoring of the network, without the need for extra probing-packets. There are two problems that must be addressed to allow monitoring the network using data that is transmitted anyway. In the current version of RealSim's network survey application the radio is always on and only sends broadcast packets. This is a rather unusual usage scheme. In a real use case there is likely to be some Radio Duty Cycling (RDC) protocol in place and most packets will be unicast. The effect of a packet not being received due to RDC is covered by the simulator and must not be accounted for when monitoring the network. Thus packet loss due to a turned off radio must be eliminated when generating the network survey. Also, ideally the RDC protocol will send a unicast packet when the receiver of the target node just woke up and is listening. This however means that other neighboring nodes might never hear from that node, because none of the packets it sends are addressed to them and their radio is turned off when packets are sent to other nodes. In consequence connections that actually exist are not captured.

How to mitigate these effects, possibly by sending broadcast messages in regular intervals or correlating connections and estimating a connection based on the current state of other connections, requires further research. A more comprehensive discussion can be found in section 3.8.5, page 77.

Future Work: Clock Drift An aspect that was not looked into was clock drift. Especially the internal clock used by the TMote Sky WSN nodes is prone to temperature changes. Contiki does calibrate the internal clock against clock crystal at startup, but does no readjustment at runtime. For this work it played a subordinate role, because the experiments ran for short durations.

Support for clock drift was added to Cooja in a more recent version than used in this work (`9261ff5d` [Con]). To add clock drift to the simulation, the RealSim client must be extended to cover this metric, too. For the TMote Sky this might be implemented by first calibrating the clock crystal to a reference clock, and then using the clock crystal to monitor the internal clock.

7.2.2 Simulator

A major part of the divergences between simulation and testbed can currently be accounted to the simulator. Thus the feasibility of the presented approach heavily relies on the availability and quality of a suitable simulator. Most notably there must be simulation support for the hardware. Unfortunately even the implementation of the Tmote Sky node in Cooja/MSPSim Simulator, which is probably the most widely used ILS simulation model in the WSN domain, is incomplete.

Future Work: Noise Model One of the issues faced during this work was the lack of noise models for WSN. While I did manage to sample the background noise at a high rate, this was only possible by transmitting the data via the serial interface. Collecting the data using IEEE 802.15.4 is not possible due to the limited bandwidth and buffering capacities. Although there has been some research in this direction, this was driven by a naive approach: Tracing the noise and using the collected data to create a model (e.g. [LCL07]). In my opinion a theoretical analysis of the characteristics of the protocols used in the 2.4 GHz band and then trying to model and detect their characteristics is more expedient. This is discussed in detail in section 3.7, page 63.

Future Work: Improve Simulators Currently Cooja, including its two ILSs MSPSim and Avrora, is the only suitable simulator for this approach. With RealSim/DryRun I created tools that allow achieving simulation results that can be compared to experiments from the testbed. This allowed me to uncover multiple bugs. Two of those effecting the simulation results in the evaluation are still open.

The first bug is that the capture effect is not implemented in Coojas radio models. This describes the ability to successfully receive the strongest of multiple interfering signals. Two situations can arise: Stronger-first, where the signal that is already being received is not interfered by a weaker one and stronger-last, where a weak signal is being received and suppressed by a stronger signal. Especially stronger-last depends on the hardware used. For example IEEE 802.15.4 radios can detect that they lost a signal. However, the literature and documentation is unclear under which circumstances the new packet can be received (see section 2.6.7.3, page 35).

The second bug is the incorrect implementation of the Clear Channel Assessment (CCA) mode of the radio chip used. Currently, the simulated radio chip takes only the signal strength into account when deciding whether the channel is in use. By default, the real radio chip however also considers the channel in use when it detects a valid signal. Unfortunately the requirements for the radio chip to successfully detect a signal are not known and must be investigated (see also section 6.5.2, page 110).

Once these issues are solved, it is very likely that further bugs can be uncovered by consequently testing radio-related configuration parameters and comparing their effect on testbed and simulation.

7.2.3 Optimization

Currently the DryRun tool chain only provides running an experiment campaign, which scans a complete parameter-space. Due to the exponential growth of the parameter space, four or five parameters can easily be sufficient to create a parameter space with over 100 000 data points. Especially as the simulations are Monte-Carlo simulations and must therefore be repeated multiple times. Such huge parameter spaces can only be addressed using heuristics and/or machine learning.

While there are multiple machine learning tools most of them cannot be used out-of-the box and require use case specific adjustments. The reasons are discussed in section 3.3, page 44.

7.2.4 Miscellaneous

While investigating the cause for deviations between simulation and testbed, it was mandatory to also look into the workings of the lower network layers. Improving these, however, is not within the scope of this work.

Dynamic CCA for ContikiMAC While evaluating the effect of the CCA threshold on the RDC protocols it showed that a ContikiMAC is especially vulnerable to a bad configuration (section 6.5, page 110). By dynamically adjusting the CCA threshold to a value just above the background noise the performance of ContikiMAC can likely be improved significantly. There are many ways how this can be done. One option would be to sample the background noise and use some variation of an I-controller. Taking the signal strength of received packets into account might improve results, too. Another idea that might be worth investigating is to use separate CCA thresholds for transmission and reception. DryRun and RealSim provide the perfect tools to improve and thoroughly test protocols like ContikiMAC without having to run hundreds of experiments on real hardware.

Bibliography

[AK04] J. Al-Karaki and A. Kamal. "Routing Techniques in Wireless Sensor Networks: A Survey". In: *IEEE Wireless Communications* 6 (2004), pp. 6–28. ISSN: 1536-1284. DOI: 10.1109/MWC.2004.1368893. (Cit. on p. 41).

[AKO14] A. AlSayyari, I. Kostanic, and C. E. Otero. "An Empirical Path Loss Model for Wireless Sensor Network Deployment in an Artificial Turf Environment". In: *Proceedings of the 11th IEEE International Conference on Networking, Sensing and Control*. IEEE, 2014, pp. 637–642. ISBN: 978-1-4799-3106-4. DOI: 10.1109/ICNSC.2014.6819700. (Cit. on p. 73).

[Aky+02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "Wireless Sensor Networks: A Survey". In: *Elsevier Computer Networks* 4 (2002), pp. 393–422. DOI: 10.1016/S1389-1286(01)00302-4 (cit. on pp. vii, 1 sq., 7).

[Ami+13] B. Aminian, J. Araujo, M. Johansson, and K. H. Johansson. "GISOO: A Virtual Testbed for Wireless Cyber-Physical Systems". In: *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society (IECON 2013)*. IEEE, 2013, pp. 5588–5593. ISBN: 978-1-4799-0224-8. DOI: 10.1109/IECON.2013.6700049. (Cit. on pp. 22, 53, 60).

[Asc+10] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn. "BonnMotion: A Mobility Scenario Generation and Analysis Tool". In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST, 2010, p. 51. ISBN: 978-963-9799-87-5. DOI: 10.4108/ICST.SIMUTOOLS2010.8684. (Cit. on p. 34).

[Atm11] Atmel Corporation. *ATmega128/L Datasheet (2467X–AVR–06/11)*. 2011 (cit. on pp. 1, 12).

[Atm14] Atmel Corporation. *Lightweight Mesh Developer Guide*. 2014 (cit. on p. 43).

[Avr] *Avrora*. URL: http://sourceforge.net/projects/avrora/ (visited on 12/10/2014) (cit. on pp. 21, 84).

[avra] *ATMEL AVR Simulator*. URL: http://avr.sourceforge.net/ (visited on 12/11/2014) (cit. on p. 21).

[avrb] *Atmel® Studio*. URL: http://www.atmel.com/atmelstudio (visited on 12/11/2014) (cit. on p. 21).

[avrc] *avremu*. URL: https://gitlab.brokenpipe.de/stettberger/avremu/tree/master (visited on 12/11/2014) (cit. on p. 21).

[avrd] *Labcenter Electronics : Proteus VSM - SPICE Simulator/Debugger for Atmel AVR*. URL: http://www.labcenter.com/products/vsm/avr.cfm (visited on 12/11/2014) (cit. on p. 21).

[avre] *mjsim - Simulator for AVR assembly code and emulator for the MeggyJr*. URL: https://code.google.com/p/mjsim/ (visited on 12/11/2014) (cit. on p. 21).

[avrf] *simavr*. URL: https://gitorious.org/simavr (visited on 12/11/2014) (cit. on p. 21).

[avrg] *Simulavr homepage*. URL: http://www.nongnu.org/simulavr/ (visited on 12/11/2014) (cit. on p. 21).

[avrh] *VMLAB for AVR microcontrollers*. URL: http://www.amctools.com/vmlab.htm (visited on 12/11/2014) (cit. on p. 21).

[Bac+13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. Schmidt. "RIOT OS: Towards an OS for the Internet of Things". In: *Proceedings of the IEEE Conference on Computer Communications, Workshops (INFOCOM WKSHPS 2013)*. IEEE, 2013, pp. 79–80. ISBN: 978-1-4799-0056-5. DOI: 10.1109/INFCOMW.2013.6970748. (Cit. on p. 42).

[Ber+10] L. Bergamini, C. Crociani, A. Vitaletti, and M. Nati. "Validation of WSN Simulators Through a Comparison with a Real Testbed." In: *Proceedings of the 7th ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Nnetworks (PE-WASUN 2010)*. ACM, 2010, pp. 103–104. ISBN: 978-1-4503-0276-0. DOI: 10.1145/1868589.1868611. (Cit. on pp. 61, 73).

[Beu+09] J. Beutel, K. Römer, M. Ringwald, and M. Woehrle. "Deployment Techniques for Sensor Networks". In: *Sensor Networks*. Signals and Communication Technology. Springer, 2009. Chap. Deployment, pp. 219–248. ISBN: 978-3-642-01340-9. DOI: 10.1007/978-3-642-01341-6_9. (Cit. on pp. 2, 9).

[BFG14] M. Bacco, E. Ferro, and A. Gotta. "UAVs in WSNs for Agricultural Applications: An Analysis of the Two-Ray Radio Propagation Model". In: *Proceedings of IEEE SENSORS 2014*. IEEE, 2014, pp. 130–133. ISBN: 978-1-4799-0162-3. DOI: 10.1109/ICSENS.2014.6984950. (Cit. on p. 32).

[BKR03] J. Beutel, O. Kasten, and M. Ringwald. "Poster Abstract: BTnodes - a Distributed Platform for Sensor Nodes". In: *Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, 2003, p. 292. ISBN: 1581137079. DOI: 10.1145/958491.958526. (Cit. on p. 51).

[Boa+09] C. A. Boano, Z. He, Y. Li, T. Voigt, M. Zuniga, and A. Willig. "Controllable Radio Interference for Experimental and Testing Purposes in Wireless Sensor Networks". In: *Proceedings of the 34th IEEE Conference on Local Computer Networks (LCN 2009)*. IEEE, 2009, pp. 865–872. ISBN: 978-1-4244-4488-5. DOI: 10.1109/LCN.2009.5355013. (Cit. on pp. 29, 60).

[Boa+10a] C. A. Boano, J. Brown, Z. He, U. Roedig, and T. Voigt. "Low-Power Radio Communication in Industrial Outdoor Deployments: The Impact of Weather Conditions and ATEX-Compliance". In: *Sensor Applications, Experimentation, and Logistics*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2010, pp. 159–176. ISBN: 978-3-642-11870-8_11. DOI: 10.1007/978-3-642-11870-8_11. (Cit. on pp. 18, 61).

[Boa+10b] C. A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig. "The Impact of Temperature on Outdoor Industrial Sensornet Applications". In: *IEEE Transactions on Industrial Informatics* 3 (2010), pp. 451–459. ISSN: 1551-3203. DOI: 10.1109/TII.2009.2035111. (Cit. on pp. 18, 61).

[Boa+11] C. A. Boano, K. Römer, F. Österlind, and T. Voigt. "Demo Abstract: Realistic Simulation of Radio Interference in COOJA". In: *European Conference on Wireless Sensor Networks (EWSN 2011)*. 2011, p. 2. (Cit. on p. 29).

[Bou11] A. Boulis. *Castalia User's Manual, Version 3.2*. 2011. URL: https://forge.nicta.com.au/docman/view.php/301/592/Castalia+-+User+Manual.pdf (cit. on p. 34).

[BS09] S. Brown and C. J. Sreenan. "Software Update Recovery for Wireless Sensor Networks". In: *Proceedings of International Conference on Sensor Networks Applications, Experimentation and Logistics (SENSAPPEAL 2009)*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2009, pp. 107–125. ISBN: 3642118690. DOI: 10.1007/978-3-642-11870-8_8. (Cit. on p. 41).

[Bue+06] M. Buettner, G. V. Yee, E. Anderson, and R. Han. "X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks". In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006)*. ACM Press, 2006, p. 307. ISBN: 1595933433. DOI: 10.1145/1182807.1182838. (Cit. on p. 110).

[Bur+09] R. Burchfield, E. Nourbakhsh, J. Dix, K. Sahu, S. Venkatesan, and R. Prakash. "RF in the Jungle: Effect of Environment Assumptions on Wireless Experiment Repeatability". In: *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–6. DOI: 10.1109/ICC.2009.5199421. (Cit. on p. 61).

[CA09] K. R. Chowdhury and I. F. Akyildiz. "Interferer Classification, Channel Selection and Transmission Adaptation for Wireless Sensor Networks". In: *2009 IEEE International Conference on Communications (ICC 2009)*. IEEE, 2009, pp. 1–5. DOI: 10.1109/ICC.2009.5199098. (Cit. on p. 25).

[Car+14] R. C. Carrano, D. Passos, L. C. S. Magalhaes, and C. V. N. Albuquerque. "Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks". In: *IEEE Communications Surveys & Tutorials* 1 (2014), pp. 181–194. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.052213.00116. (Cit. on p. 56).

[cas] *Castalia*. URL: https://castalia.forge.nicta.com.au/ (visited on 12/11/2014) (cit. on p. 21).

[Cat13] Catverine. *2-ray ground reflected Power vs distance.jpg - Wikimedia Commons*. 2013. URL: https://commons.wikimedia.org/wiki/File:2-ray%7B%5C_%7Dground%7B%5C_%7Dreflected%7B%5C_%7DPower%7B%5C_%7Dvs%7B%5C_%7Ddistance.jpg (cit. on p. 33).

[Cer+10] M. Ceriotti, M. Chini, A. L. Murphy, G. P. Picco, F. Cagnacci, and B. Tolhurst. "Motes in the Jungle: Lessons Learned from a Short-Term WSN Deployment in the Ecuador Cloud Forest". In: *Proceedings of the 4th International Workshop on Real-World Wireless Sensor Networks (REALWSN 2010)*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 25–36. ISBN: 978-3-642-17519-0, 978-3-642-17520-6. DOI: 10.1007/978-3-642-17520-6_3. (Cit. on pp. 2, 53).

[CFP] *Call for Papers: European Conference on Wireless Sensor Networks (EWSN 2015)*. 2015. URL: http://www.cister.isep.ipp.pt/ewsn2015/cfp (cit. on p. 7).

[Cha10] T. M. Chan. "Comparison-Based Time-Space Lower Bounds for Selection". In: *ACM Transactions on Algorithms* 2 (2010), pp. 1–16. ISSN: 15496325. DOI: 10.1145/1721837.1721842. (Cit. on p. 50).

[Chi+11] M. Chini, M. Ceriotti, R. Marfievici, A. L. Murphy, and G. P. Picco. "Demo: TRIDENT, Untethered Observation of Physical Communication Made to Share". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys 2011)*. SenSys '11. ACM, 2011, pp. 409–410. ISBN: 978-1-4503-0718-5. DOI: 10.1145/2070942.2071014. (Cit. on p. 83).

[Con] *Contiki Git Repository*. URL: https://github.com/contiki-os/contiki/ (cit. on pp. 58, 80, 82–85, 94, 107, 112, 114, 118, 129).

[con15] W. contributors. *Two-ray ground-reflection model*. 2015. URL: http://en.wikipedia.org/w/index.php?title=Two-ray%7B%5C_%7Dground-reflection%7B%5C_%7Dmodel%7B%5C&%7Doldid=612785061 (visited on 03/10/2015) (cit. on p. 32).

[Cooa] *CoojaTrace Git Repository*. URL: https://github.com/cmorty/coojatrace/ (cit. on p. 94).

[Coob] *CoojaTrace Plot Git Repository*. URL: https://github.com/cmorty/coojatraceplot (cit. on p. 95).

[Cooc] *CoojaTrace Sqlite Git Repository*. URL: https://github.com/cmorty/coojatracesqlite (cit. on p. 95).

[Cou+12] G. Coulson, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, P. Hurni, M. Anwander, G. Wagenknecht, S. P. Fekete, A. Kröller, and T. Baumgartner. "Flexible Experimentation in Wireless Sensor Networks". In: *Communications of the ACM* 1 (2012), pp. 82–90. ISSN: 0001-0782. DOI: 10.1145/2063176.2063198. (Cit. on pp. 22, 95, 108).

[Cro04] I. Crossbow Technology. *MPR / MIB User's Manual*. 2004 (cit. on p. 15).

[CT10] Y. Chen and A. Terzis. "On the Mechanisms and Effects of Calibrating RSSI Measurements for 802.15.4 Radios". In: *Proceedings of the 7th European conference on Wireless Sensor Networks (EWSN 2010)*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 256–271. ISBN: 978-3-642-11916-3. DOI: 10.1007/978-3-642-11917-0. (Cit. on pp. 30, 36, 75).

[Cul+02] D. Culler, J. Hill, M. Horton, K. Pister, R. Szewczyk, and A. Wood. *MICA: The Commercialization of Microsensor Motes*. Tech. rep. 2002. URL: http://www.sensorsmag.com/networking-communications/mica-the-commercialization-microsensor-motes-1070 (cit. on p. 7).

[Dat] *Data Extractor Git Repository*. URL: https://github.com/cmorty/dataextractor (cit. on pp. 96, 128).

[Dez+14a] B. Dezfouli, M. Radi, S. A. Razak, T. Hwee-Pink, and K. A. Bakar. "Modeling Low-Power Wireless Communications". In: *Journal of Network and Computer Applications* (2014). ISSN: 10848045. DOI: 10.1016/j.jnca.2014.02.009. (Cit. on p. 36).

[Dez+14b] B. Dezfouli, M. Radi, K. Whitehouse, S. A. Razak, and H.-P. Tan. "CAMA: Efficient Modeling of the Capture Effect for Low-Power Wireless Networks". In: *ACM Transactions on Sensor Networks (TOSN)* 1 (2014), pp. 1–43. ISSN: 15504859. DOI: 10.1145/2629352. (Cit. on pp. 36, 112).

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors". In: *29th Annual IEEE International Conference on Local Computer Networks*. IEEE (Comput. Soc.), 2004, pp. 455–462. ISBN: 0-7695-2260-2. DOI: 10.1109/LCN.2004.38. (Cit. on pp. 3, 42).

[Dre+16] F. Dressler, S. Ripperger, M. Hierold, T. Nowak, C. Eibel, B. Cassens, F. Mayer, K. Meyer-Wegener, and A. Kolpin. "From Radio Telemetry to Ultra-Low-Power Sensor Networks: Tracking Bats in the Wild". In: *IEEE Communications Magazine* 1 (2016), pp. 129–135. ISSN: 0163-6804. DOI: 10.1109/MCOM.2016.7378438. (Cit. on p. 9).

[Dry] *Dryrun Git Repository*. URL: https://github.com/cmorty/dryrun (cit. on pp. 91, 128).

[Dun+07] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. "Software-Vased On-Line Energy Estimation for Sensor Nodes". In: *Proceedings of the 4th workshop on Embedded networked sensors (EmNets 2007)*. ACM, 2007, pp. 28–32. ISBN: 978-1-59593-694-3. DOI: 10.1145/1278972.1278979. (Cit. on pp. 62, 94).

[Dun11] A. Dunkels. *The Contikimac Radio Duty Cycling Protocol*. Tech. rep. Swedish Institute of Computer Science, 2011. URL: http://soda.swedish-ict.se/5128/%20http://soda.swedish-ict.se/5128/1/contikimac-report.pdf (cit. on pp. 19, 110).

[Dur+12] M. Durisic, Z. Tafa, G. Dimic, and V. Milutinovic. "A Survey of Military Applications of Wireless Sensor Networks". In: *2012 Mediterranean Conference on Embedded Computing (MECO 2012)*. 2012, pp. 196–199. ISBN: 978-9940-9436-0-8 (cit. on p. 11).

[Eri+09] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. "COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks". In: *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques (Simutools 2009)*. ICST, 2009, 27:1–27:7. ISBN: 978-963-9799-45-5. DOI: 10.4108/ICST.SIMUTOOLS2009.5637. (Cit. on p. 22).

[Ert+06] E. Ertin, M. Nesterenko, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao. "Kansei: A Testbed for Sensing at Scale". In: *Proceedings of the fifth international conference on Information processing in sensor networks (IPSN 2006)*. ACM Press, 2006, pp. 399–406. ISBN: 1595933344. DOI: 10.1145/1127777.1127838. (Cit. on pp. 18, 22).

[Eve13] Everspin Technologies Inc. *Comparing Technologies: MRAM vs. FRAM*. Tech. rep. 2013, pp. 1–7. URL: http://www.everspin.com/file/227/download (cit. on p. 14).

[Flo11] L. Florian. "Entwicklung eines flexiblen Frameworks zur Erfassung von Metriken bei der Simulation von Sensornetzwerken". Bachelor Theses. Uni Erlangen-Nürnberg, 2011, pp. 1–68 (cit. on p. 94).

[FMT12] F. Ferrari, A. Meier, and L. Thiele. "Accurate Clock Models for Simulating Wireless Sensor Networks". In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST, 2012, p. 21. ISBN: 978-963-9799-87-5. DOI: 10.4108/ICST.SIMUTOOLS2010.8693. (Cit. on pp. 58 sq.).

[For00] R. Forster. "Manchester Encoding: Opposing Definitions Resolved". In: *Engineering Science & Education Journal* 6 (2000), pp. 278–280. ISSN: 0963-7346. DOI: 10.1049/esej:20000609. (Cit. on p. 27).

[Gan+09] S. Ganeriwal, I. Tsigkogiannis, H. Shim, V. Tsiatsis, M. Srivastava, and D. Ganesan. "Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks". In: *IEEE/ACM Transactions on Networking* 3 (2009), pp. 843–856. ISSN: 1063-6692. DOI: 10.1109/TNET.2008.2001953 (cit. on p. 59).

[Gar+11] K. Garg, A. Förster, D. Puccinelli, and S. Giordano. "Towards Realistic and Credible Wireless Sensor Network Evaluation". In: *Proceedings of the 3rd International ICST Ad Hoc Networks (ADHOCNETS 2011)*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2011, pp. 49–64. ISBN: 9783642290954. DOI: 10.1007/978-3-642-29096-1_4. (Cit. on p. 73).

[Gar14] Gartner Inc. *Press Release: Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business*. Tech. rep. 2014. URL: https://www.gartner.com/newsroom/id/2819918 (cit. on p. 2).

[GCM11] O. Gama, P. Carvalho, and P. M. Mendes. "Modelling the Impact of Software Components on Wireless Sensor Network Performance". In: *1st Portuguese Conference on Sensor Networks*. 2011. ISBN: 978-989-96001-4-0. (Cit. on pp. 57, 73).

[GHR13] V. C. Gungor, G. P. Hancke, and V. a. R. G. N. R. *Industrial Wireless Sensor Networks: Applications, Protocols, and Standards*. Auflage: N. Routledge Chapman & Hall, 2013, p. 374. ISBN: 9781466500518. (Cit. on p. 16).

[Git16] GitHub Inc. *Understanding the GitHub Flow · GitHub Guides*. 2016. URL: https://guides.github.com/introduction/flow/ (visited on 02/25/2016) (cit. on p. 118).

[GJP10] C. Guo, M. Jacobsson, and R. V. Prasad. "A Case Study of Networked Sensors by Simulations and Experiments". In: *Proceedings of the 11th International Conference on Thermal, Mechanical Multi-Physics Simulation, and Experiments in Microelectronics and Microsystems (EuroSimE 2010)*. 2010, pp. 1–5. DOI: 10.1109/ESIME.2010.5464517 (cit. on pp. 61, 73).

[Gna+09] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. "Collection Tree Protocol". In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*. ACM Press, 2009, pp. 1–14. ISBN: 9781605585192. DOI: 10.1145/1644038.1644040. (Cit. on p. 19).

[Goe+14] G. Goel, S. H. Melvin, Y. Lostanlen, and D. Hatzinakos. "Connectivity Analysis of Indoor Wireless Sensor Networks Using Realistic Propagation Models". In: *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems (MSWiM 2014)*. ACM Press, 2014, pp. 13–20. ISBN: 9781450330305. DOI: 10.1145/2641798.2641803. (Cit. on pp. 60, 74).

[Goo14] T. Goodspeed. "Phantom Boundaries and Cross-Layer Illusions in 802.15.4 Digital Radio". In: *2014 IEEE Security and Privacy Workshops (SPW 2014)*. IEEE, 2014, pp. 181–184. ISBN: 978-1-4799-5103-1. DOI: 10.1109/SPW.2014.33. (Cit. on p. 28).

[Goy+10] M. Goyal, S. Prakash, W. Xie, Y. Bashir, H. Hosseini, and A. Durresi. "Evaluating the Impact of Signal to Noise Ratio on IEEE 802.15.4 PHY-Level Packet Loss Rate". In: *Proceedings of the 13th International Conference on Network-Based Information Systems*. IEEE, 2010, pp. 279–284. ISBN: 978-1-4244-8053-1. DOI: 10.1109/NBiS.2010.97. (Cit. on p. 28).

[Hab+13] R. Haber, A. Peter, C. E. Otero, I. Kostanic, and A. Ejnioui. "A Support Vector Machine for Terrain Classification in On-Demand Deployments of Wireless Wensor Networks". In: *Proceedings of the IEEE International Systems Conference (SysCon 2013)*. IEEE, 2013, pp. 841–846. ISBN: 978-1-4673-3108-1. DOI: 10.1109/SysCon.2013.6549982. (Cit. on p. 73).

[HB09] P. Hurni and T. Braun. "Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments". In: *8th International IFIP-TC 6 Networking Conference*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 1–13. ISBN: 978-3-642-01398-0, 978-3-642-01399-7. DOI: 10.1007/978-3-642-01399-7_1. (Cit. on p. 73).

[HC04] J. W. Hui and D. Culler. "The Dynamic Vehavior of a Data Dissemination Protocol for Network Programming at Scale". In: *Proceedings of the 2nd Iternational Conference on Embedded Networked Sensor Systems (SenSys 2004)*. ACM Press, 2004, p. 81. ISBN: 1581138792. DOI: 10.1145/1031495.1031506. (Cit. on pp. 19, 41).

[He+12] D. He, G. Mujica, J. Portilla, and T. Riesgo. "Simulation Tool and Case Study for Planning Wireless Sensor Network". In: *Proceedings of the 38th Annual Conference on IEEE Industrial Electronics Society (IECON 2012)*. 2012, pp. 6024–6028. DOI: 10.1109/IECON.2012.6389097. (Cit. on p. 73).

[HIT08] A. Hasler, C. Igor Talzi, and S. G. Tschudin. "Wireless Sensor Networks in Permafrost Rresearch - Concept, Requirements, Implementation and Challenges". In: *Proceedings of the 9th Internationall Conference on Permafrost (NICOP 2008)*. 2008, pp. 669–674. (Cit. on pp. 53, 59, 61).

[HL10] G. P. Halkes and K. G. Langendoen. "Experimental Evaluation of Simulation Abstractions for Wireless Sensor Network MAC Protocols". In: *EURASIP Journal on Wireless Communications and Networking* (2010), 24:1–24:10. ISSN: 2378-4865. DOI: 10.1109/CAMAD.2009.5161468. (Cit. on pp. 34, 36, 60 sq.).

[HNL08] M. Healy, T. Newe, and E. Lewis. "Analysis of Hardware Encryption Versus Software Encryption on Wireless Sensor Network Motes". In: *Lecture Notes in Electrical Engineering* (2008), pp. 3–14. ISSN: 18761100. DOI: 10.1007/978-3-540-79590-2-1 (cit. on p. 31).

[Hön+14] T. Hönig, H. Janker, O. Mihelic, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. "Proactive Energy-Aware Programming with PEEK". In: *Proceedings of the 2014 Conference on Timely Results in Operating Systems (TRIOS 2014)*. USENIX Association, 2014, pp. 1–14. (Cit. on p. 50).

[HS36] S. Hecht and S. Shlaer. "INTERMITTENT STIMULATION BY LIGHT: V. The Relation between Intensity and Critical Frequency for Different Parts of the Spectrum". In: *The Journal of General Physiology* 6 (1936), pp. 965–977. ISSN: 0022-1295. DOI: 10.1085/jgp.19.6.965. (Cit. on p. 59).

[Huo+09] H. Huo, Y. Xu, C. Bilen, and H. Zhang. "Coexistence Issues of 2.4GHz Sensor Networks with Other RF Devices at Home". In: 2009, pp. 200–205. DOI: 10.1109/SENSORCOMM.2009.40. (Cit. on p. 18).

[Hur+11] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder. "On the Accuracy of Software-Based Energy Estimation Techniques". In: *Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN 2011)*. Springer, 2011, pp. 49–64. ISBN: 978-3-642-19185-5. DOI: 10.1007/978-3-642-19186-2_4. (Cit. on pp. 50, 62).

[HWS12] C. Haas, J. Wilke, and V. Stöhr. "Realistic Simulation of Energy Consumption in Wireless Sensor Networks". In: *Wireless Sensor Networks*. LNCS 7158. Springer, 2012, pp. 82–97. ISBN: 978-3-642-28168-6. DOI: 10.1007/978-3-642-28169-3_6. (Cit. on pp. 50, 62).

[IEE11] IEEE P802.15 Working Group. *IEEE 802.15.4-2011 specification*. September. The Institute of Electrical and Electronics Engineers, Inc, 2011, p. 314. ISBN: 978-0-7381-6684-1 (cit. on p. 30).

[Ist+14] T. Istomin, R. Marfievici, A. L. Murphy, and G. P. Picco. "TRIDENT: In-field Connectivity Assessment for Wireless Sensor Networks". In: *Proceedins of the 7th Extreme Conference on Communication and Computing (ExtremeCom 2014)*. 2014, pp. 1–6 (cit. on p. 83).

[JR13] M. Jacobsson and C. Rohner. "Comparing Wireless Flooding Protocols Using Trace-Based Simulations". In: *EURASIP Journal on Wireless Communications and Networking* 1 (2013), p. 169. ISSN: 1687-1499. DOI: 10.1186/1687-1499-2013-169. (Cit. on p. 73).

[KCC13] A. Kamthe, M. Á. Carreira-Perpiñán, and A. E. Cerpa. "Improving Wireless Link Simulation using Multilevel Markov Models". In: *ACM Transactions on Sensor Networks* 1 (2013), pp. 1–28. ISSN: 15504859. DOI: 10.1145/2529991. (Cit. on pp. 35, 73).

[Kel+11] M. Keller, M. Woehrle, R. Lim, J. Beutel, and L. Thiele. "Comparative Performance Analysis of the PermaDozer Protocol in Diverse Deployments". In: *Proceedings of the 36th Conference on Local Computer Networks*. IEEE, 2011, pp. 957–965. ISBN: 978-1-61284-928-7. DOI: 10.1109/LCN.2011.6115578. (Cit. on p. 59).

[Ker+10] F. Kerasiotis, A. Prayati, C. Antonopoulos, C. Koulamas, and G. Papadopoulos. "Battery Lifetime Prediction Model for a WSN Platform". In: *Proceedings of the 4th International Conference on Sensor Technologies and Applications*. IEEE, 2010, pp. 525–530. ISBN: 978-1-4244-7538-4. DOI: 10.1109/SENSORCOMM.2010.85. (Cit. on p. 62).

[KKP99] J. M. Kahn, R. H. Katz, and K. S. J. Pister. "Next century challenges: Mobile Networking for "Smart Dust"". In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999)*. ACM Press, 1999, pp. 271–278. ISBN: 1581131429. DOI: 10.1145/313451.313558. (Cit. on pp. 2, 11).

[Ko+11] J. G. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler. "Industry: Beyond Interoperability: Pushing the Performance of Sensor Metwork IP Stacks". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys 2011)*. SenSys '11. ACM Press, 2011, pp. 1–11. ISBN: 978-1-4503-0718-5. DOI: 10.1145/2070942.2070944. (Cit. on p. 16).

[Ko+12] J. Ko, K. Klues, C. Richter, W. Hofer, B. Kusy, M. Bruenig, T. Schmid, Q. Wang, P. Dutta, and A. Terzis. "Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone)". In: *Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN 2012)*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 98–114. ISBN: 978-3-642-28168-6. DOI: 10.1007/978-3-642-28169-3. (Cit. on p. 11).

[Kot+04] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. "Experimental Evaluation of Wireless Simulation Assumptions". In: *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004)*. ACM, 2004, pp. 78–82. ISBN: 1-58113-953-5. DOI: 10.1145/1023663.1023679. (Cit. on p. 61).

[KSC08] Y. Kim, H. Shin, and H. Cha. "Y-MAC: An Energy-Efficient Multi-channel MAC Protocol for Dense Wireless Sensor Networks". In: *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN 2008)*. IEEE, 2008, pp. 53–63. ISBN: 978-0-7695-3157-1. DOI: 10.1109/IPSN.2008.27. (Cit. on p. 25).

[KSM08] A. Kiryushin, A. Sadkov, and A. Mainwaring. "Real-World Performance of Clear Channel Assessment in 802.15.4 Wireless Sensor Networks". In: *Proceedings of the 2nd International Conference on Sensor Technologies and Applications (SENSORCOMM 2008)*. IEEE, 2008, pp. 625–630. ISBN: 978-0-7695-3330-8. DOI: 10.1109/SENSORCOMM.2008.58. (Cit. on pp. 36, 56).

[KVV11] E. Kolega, V. Vescoukis, and D. Voutos. "Assessment of network simulators for real world WSNs in forest environments". In: *2011 IEEE International Conference on Networking, Sensing and Control (ICNSC 2011)*. 2011, pp. 427–432. DOI: 10.1109/ICNSC.2011.5874918. (Cit. on p. 73).

[KW07] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. 1st ed. Wiley-Interscience, 2007. ISBN: 9780470519233. (Cit. on p. 19).

[LC14] T. Liu and A. E. Cerpa. "Data-Driven Link Quality Prediction Using Link Features". In: *ACM Transactions on Sensor Networks* 2 (2014), pp. 1–35. ISSN: 15504859. DOI: 10.1145/2530535. (Cit. on p. 16).

[LCL07] H. Lee, A. Cerpa, and P. Levis. "Improving Wireless Simulation Through Noise Modeling". In: *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks (IPSN 2007)*. IEEE, 2007, pp. 21–30. ISBN: 978-1-59593-638-7. DOI: 10.1109/IPSN.2007.4379661. (Cit. on pp. 12, 29, 60, 72, 75, 129).

[Lee06] J.-S. Lee. "Performance Evaluation of IEEE 802.15.4 for Low-Rate Wireless Personal Area Networks". In: *IEEE Transactions on Consumer Electronics* 3 (2006), pp. 742–749. ISSN: 0098-3063. DOI: 10.1109/TCE.2006.1706465. (Cit. on p. 75).

[Lev+03] P. Levis, N. Lee, M. Welsh, and D. Culler. "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications". In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, 2003, pp. 126–137. ISBN: 1-58113-707-9. DOI: 10.1145/958491.958506. (Cit. on p. 22).

[Lev+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. "TinyOS: An Operating System for Sensor Networks". In: *Ambient Intelligence*. Springer, 2005, pp. 115–148. ISBN: 978-3-540-27139-0. DOI: 10.1007/3-540-27139-2_7. (Cit. on pp. 22, 42, 45).

[Li+10] Q. Li, F. Österlind, T. Voigt, S. Fischer, and D. Pfisterer. "Making Wireless Sensor Network Simulators Cooperate". In: *Proceedings of the 7th ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Nnetworks (PE-WASUN 2010)*. ACM Press, 2010, pp. 95–98. ISBN: 9781450302760. DOI: 10.1145/1868589.1868607. (Cit. on p. 61).

[Liu+10] Liu Feng, Xu Jinwu, Yang Jianhong, and Li Min. "Noise Recognition of Industrial Wireless Sensor Networks Based on Fuzzy Controller". In: *2010 International Conference on Information, Networking and Automation (ICINA 2010)*. IEEE, 2010, pp. V1–457–V1–461. ISBN: 978-1-4244-8104-0. DOI: 10.1109/ICINA.2010.5636518. (Cit. on p. 35).

[log] *Log4j – Log4j 2 Guide - Apache Log4j 2*. URL: http://logging.apache.org/log4j/2.x/ (visited on 10/28/2015) (cit. on p. 94).

[Loh09] D. Lohmann. "Aspect-Awareness in the Development of Configurable System Software". PhD thesis. Friedrich-Alexander Universit{ä}t Erlangen-N{ü}rnberg, 2009. URL: http://www4.informatik.uni-erlangen.de/Publications/2009/lohmann%7B%5C_%7D09%7B%5C_%7Ddiss.pdf (cit. on p. 45).

[LRO09] A. Lalomia, G. L. Re, and M. Ortolani. "A Hybrid Framework for Soft Real-Time WSN Simulation". In: *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009)*. IEEE, 2009, pp. 201–207. ISBN: 978-0-7695-3868-6. DOI: 10.1109/DS-RT.2009.30. (Cit. on p. 22).

[LW09] J. Lu and K. Whitehouse. "Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks". In: *Proceedings of the 28th Conference on Computer Communications (INFOCOM 2009)*. IEEE, 2009, pp. 2491–2499. ISBN: 978-1-4244-3512-8. DOI: 10.1109/INFCOM.2009.5062177. (Cit. on p. 36).

[Mar+04] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. "The Flooding Time Synchronization Protocol". In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*. ACM Press, 2004, pp. 39–49. ISBN: 1581138792. DOI: 10.1145/1031495.1031501. (Cit. on p. 19).

[Mar+10] A. Marchiori, L. Guo, J. Thomas, and Q. Han. "Realistic Performance Analysis of WSN Protocols Through Trace Based Simulation". In: *Proceedings of the 7th ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Nnetworks (PE-WASUN 2010)*. ACM, 2010, pp. 87–94. ISBN: 978-1-4503-0276-0. DOI: 10.1145/1868589.1868606. (Cit. on pp. 34, 73).

[Mar+13] R. Marfievici, A. L. Murphy, G. P. Picco, F. Ossi, and F. Cagnacci. "How Environmental Factors Impact Outdoor Wireless Sensor Networks: A Case Study". In: *Proceedings of the 10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2013)*. IEEE, 2013, pp. 565–573. ISBN: 978-1-4799-3408-9. DOI: 10.1109/MASS.2013.13. (Cit. on p. 61).

[max] *Microcontroller Clock—Crystal, Resonator, RC Oscillator, or Silicon Oscillator?* 2003. URL: https://www.maximintegrated.com/en/app-notes/index.mvp/id/2154 (cit. on p. 58).

[Maz+11] E. B. Mazomenos, D. De Jager, J. S. Reeve, and N. M. White. "A Two-Way Time of Flight Ranging Scheme for Wireless Sensor Networks". In: *Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN 2011)*. 2011, pp. 163–178. ISBN: 978-3-642-19185-5. DOI: 10.1007/978-3-642-19186-2_11. (Cit. on p. 76).

[mix] *MiXiM*. URL: http://mixim.sourceforge.net/ (visited on 12/11/2014) (cit. on p. 21).

[Mös+12] G. Möstl, R. Hagelauer, G. Müller, and A. Springer. "A Network and System Level Approach towards an Accurate Simulation of WSNs". In: *Computer Aided Systems Theory (EUROCAST 2011)*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2012, pp. 17–24. ISBN: 978-3-642-27578-4. DOI: 10.1007/978-3-642-27579-1_3. (Cit. on p. 73).

[Mös+13] G. Möstl, R. Hagelauer, G. Müller, and A. Springer. "STEAM-Sim". In: *Proceedings of the 8th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N 2013)*. ACM Press, 2013, pp. 61–66. ISBN: 9781450323710. DOI: 10.1145/2512840.2512849. (Cit. on pp. 22, 57).

[Mot06] Moteiv Corporation. *Moteiv: Tmote Sky Low Power Wireless Sensor Module*. 2006. URL: http://www.eecs.harvard.edu/%7B~%7Dkonrad/projects/shimmer/references/tmote-sky-datasheet.pdf (cit. on p. 34).

[MSP] *MSPSim*. URL: https://github.com/mspsim/mspsim (visited on 12/10/2014) (cit. on pp. 21, 84 sq.).

[msp] *MSP430F15x, MSP430F16x, MSP430F161x MIXED SIGNAL MICROCONTROLLER*. 2009. URL: http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf (cit. on pp. 1, 12, 58).

[ns2] *ns-2*. URL: http://nsnam.isi.edu/nsnam/index.php/Main%7B%5C_%7DPage (visited on 12/05/2014) (cit. on pp. 21, 29).

[OA06] M. Odersky and Al. *An Overview of the Scala Programming Language*. Tech. rep. Technical Report LAMP-REPORT-2006-001. EPFL, 2006, pp. 1–20. URL: http://www.scala-lang.org/docu/files/ScalaOverview.pdf (cit. on p. 92).

[Odo+13] T. O'donovan, W.-B. Pöttner, U. Roedig, J. S. Silva, R. Silva, C. J. Sreenan, V. Vassiliou, T. Voigt, L. Wolf, Z. Zinonos, J. Brown, F. Büsching, A. Cardoso, J. Cecílio, J. D. Ó, P. Furtado, P. Gil, and A. Jugel. "The GINSENG System for Wireless Monitoring and Control". In: *ACM Transactions on Sensor Networks* 1 (2013), pp. 1–40. ISSN: 15504859. DOI: 10.1145/2529975. (Cit. on p. 53).

[ÖED10] F. Österlind, J. Eriksson, and A. Dunkels. "Cooja TimeLine: A Power Visualizer for Sensor Network Simulation." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*. ACM, 2010, pp. 385–386. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1870035. (Cit. on p. 54).

[omn] *OMNeT++ Network Simulation Framework*. URL: http://www.omnetpp.org/ (visited on 12/05/2014) (cit. on pp. 21, 29).

[Öst+07] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. "Cross-level Simulation in COOJA". In: *Proceedings of teh European Conference on Wireless Sensor Networks (EWSN 2007), Poster/Demo session*. 2007. (Cit. on pp. 22, 79).

[Öst+09] F. Österlind, A. Dunkels, T. Voigt, N. Tsiftes, J. Eriksson, and N. Finne. "Sensornet Checkpointing: Enabling Repeatability in Testbeds and Realism in Simulators". In: *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN 2009)*. 2009, pp. 343–357. DOI: 10.1007/978-3-642-00224-3_22. (Cit. on p. 18).

[Ote+14] C. E. Otero, R. Haber, A. M. Peter, A. AlSayyari, and I. Kostanic. "A Wireless Sensor Networks' Analytics System for Predicting Performance in On-Demand Deployments". In: *IEEE Systems Journal* 99 (2014), pp. 1–10. ISSN: 1932-8184. DOI: 10.1109/JSYST.2014.2320324. (Cit. on p. 73).

[Pal12] A. Palm. *Application Note: Setup, Test, and Optimizing a Radio Link*. 2012 (cit. on p. 72).

[PB09] R. K. Panta and S. Bagchi. "Hermes: Fast and Energy Efficient Incremental Code Updates for Wireless Sensor Networks". In: *Proceedings of the 28th Conference on Computer Communications (INFOCOM 2009)*. IEEE, 2009, pp. 639–647. ISBN: 978-1-4244-3512-8. DOI: 10.1109/INFCOM.2009.5061971. (Cit. on p. 53).

[Per+08] E. Perla, A. Ó. Catháin, R. S. Carbajo, M. Huggard, and C. Mc Goldrick. "PowerTOSSIM z". In: *Proceedings of the 3nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N 2008)*. ACM Press, 2008, pp. 35–42. ISBN: 9781605582399. DOI: 10.1145/1454630.1454636. (Cit. on p. 62).

[Pet] PeterZegelin. *MacSim | AVR SIMULATOR FOR OSX*. URL: http://www.fracturedsoftware.com/macsimavr/ (visited on 12/11/2014) (cit. on p. 21).

[Pet+06] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. "Performance Study of IEEE 802.15.4 Using Measurements and Simulations". In: *Proceedings of the IEEE Wireless Communications and Networking Conference 2006 (WCNC 2006)*. IEEE, 2006, pp. 487–492. ISBN: 1-4244-0269-7. DOI: 10.1109/WCNC.2006.1683512. (Cit. on p. 36).

[Pet+12] P. Pettinato, N. Wirström, J. Eriksson, and T. Voigt. "Multi-Channel Two-Way Time of Flight Sensor Network Ranging". In: *Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN 2012)*. 2012, pp. 163–178. ISBN: 9783642281686. DOI: 10.1007/978-3-642-28169-3_11. (Cit. on p. 76).

[PJL02] K. Pawlikowski, H. J. Jeong, and J. R. Lee. "On Credibility of Simulation Studies of Telecommunication Networks". In: *IEEE Communications Magazine* 1 (2002), pp. 132–139. ISSN: 0163-6804. DOI: 10.1109/35.978060. (Cit. on pp. 53 sq.).

[PM88] S. K. Park and K. W. Miller. "Random Number Generators: Good Ones are Hard to Find". In: *Communications of the ACM* 10 (1988), pp. 1192–1201. ISSN: 00010782. DOI: 10.1145/63039.63042. (Cit. on p. 82).

[PPB07] H. N. Pham, D. Pediaditakis, and A. Boulis. "From Simulation to Real Deployments in WSN and Back". In: *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*. 2007, pp. 1–6. DOI: 10.1109/WOWMOM.2007.4351800 (cit. on pp. 58, 60, 73).

[PSS00] S. Park, A. Savvides, and M. B. Srivastava. "SensorSim: A Simulation Framework for Sensor Networks". In: MSWIM '00. ACM, 2000, pp. 104–111. ISBN: 1-58113-304-9. DOI: 10.1145/346855.346870. (Cit. on p. 21).

[Rap02] T. S. Rappaport. *Wireless Communications: Principles and Practice (2nd Edition)*. Prentice Hall, 2002, p. 736. ISBN: 978-0130422323. (Cit. on pp. 32 sq., 36).

[rea] *RealSim Git Repository*. URL: `https://github.com/cmorty/realsim` (cit. on pp. 34, 128).

[Ren+11] O. Rensfelt, F. Hermans, P. Gunningberg, L.-A. Larzon, and E. Bjornemo. "Repeatable Experiments with Mobile Nodes in a Relocatable WSN Testbed". In: *The Computer Journal* 12 (2011), pp. 1973–1986. ISSN: 0010-4620. DOI: `10.1093/comjnl/bxr052`. (Cit. on p. 18).

[SA05] M. K. Simon and M.-S. Alouini. *Digital Communication over Fading Channels, 2nd Edition*. 2005, p. 936. ISBN: 0471200697 (cit. on p. 36).

[Sca] *Wisebed Scala Client Git Repository*. URL: `https://github.com/cmorty/scala-client` (cit. on p. 95).

[Sie+12] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen. "How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4". In: *Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW 2012)*. IEEE, 2012, pp. 232–237. ISBN: 978-1-4673-0682-9. DOI: `10.1109/WCNCW.2012.6215496`. (Cit. on p. 12).

[Sin13] J. Sincero. "Variability Bugs in System Software". PhD thesis. Friedrich-Alexander-Universit{ä}t Erlangen-N{ü}rnberg, 2013. URL: `http://opus4.kobv.de/opus4-fau/files/3317/diss.pdf` (cit. on p. 45).

[SK11] S. Sudevalayam and P. Kulkarni. "Energy Harvesting Sensor Nodes: Survey and Implications". In: *IEEE Communications Surveys & Tutorials* 3 (2011), pp. 443–461. ISSN: 1553-877X. DOI: `10.1109/SURV.2011.060710.00094`. (Cit. on p. 16).

[SK13] O. Stecklina and A. Krumholz. "The Crux of OMNeT++ on development for a specific Wireless Sensor Node Platform, A Progress Report". In: *2. GI / ITG FACHGESPRÄCH SENSORNETZE*. 2013, pp. 21–24 (cit. on pp. 57, 63).

[SKH06] D. Son, B. Krishnamachari, and J. Heidemann. "Experimental Study of Voncurrent Transmission in Wireless Sensor Networks". In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006)*. ACM Press, 2006, pp. 237–250. ISBN: 1595933433. DOI: `10.1145/1182807.1182831`. (Cit. on p. 36).

[SLK12] M. Strübe, F. Lukas, and R. Kapitza. "Demo Abstract: CoojaTrace, Extensive Profiling for WSNs". In: *Poster and Demo Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN 2012)*. 2012, pp. 64–65. (Cit. on pp. 18, 94).

[Son+12] Z. Y. Song, M. Mostafizur, R. Mozumdar, M. Tranchero, L. Lavagno, R. Tomasi, and S. Olivieri. "Hy-Sim: Model Based Hybrid Simulation Framework for WSN Application Development". In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2012, pp. 1–8. ISBN: 978-963-9799-87-5. DOI: `10.4108/ICST.SIMUTOOLS2010.8662`. (Cit. on p. 22).

[Spo+08] M. A. Spohn, P. S. Sausen, F. Salvadori, and M. Campos. "Simulation of Blind Flooding over Wireless Sensor Networks Based on a Realistic Battery Model". In: *Proceedings of the Seventh International Conference on Networking (ICN 2008)*. IEEE, 2008, pp. 545–550. ISBN: 978-0-7695-3106-9. DOI: `10.1109/ICN.2008.100`. (Cit. on p. 62).

[Ste] *steam-sim download | SourceForge.net*. URL: `https://sourceforge.net/projects/steamsim/` (cit. on p. 22).

[Ste+12] A. Stetsko, T. Smolka, V. Matyas, and F. Jurnečka. "On the Credibility of Wireless Sensor Network Simulations: Evaluation of Intrusion Detection System". In: *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS 2012)*. SIMUTOOLS '12. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 75–84. ISBN: 978-1-4503-1510-4. (Cit. on p. 61).

[Sto+09] T. Stoyanova, F. Kerasiotis, A. Prayati, and G. Papadopoulos. "A Practical RF Propagation Model for Wireless Network Sensors". In: *Proceedings of the 3rd International Conference on Sensor Technologies and Applications*. IEEE, 2009, pp. 194–199. ISBN: 978-0-7695-3669-9. DOI: `10.1109/SENSORCOMM.2009.39`. (Cit. on pp. 32, 34).

[Sto08] I. Stojmenovic. "Simulations in Qireless Sensor and Ad Hoc Networks: Matching and Advancing Models, Mmetrics, and Solutions". In: *IEEE Communications Magazine* 12 (2008), pp. 102–107. ISSN: 0163-6804. DOI: `10.1109/MCOM.2008.4689215`. (Cit. on p. 63).

[Str+14] M. Strübe, F. Lukas, B. Li, and R. Kapitza. "DrySim: Simulation-Aided Deployment-Specific Tailoring of Mote-class WSN Software". In: *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2014)*. MSWiM '14. ACM, 2014, pp. 3–11. ISBN: 978-1-4503-3030-5. DOI: 10.1145/2641798.2641838. (Cit. on pp. 18, 53).

[Str17a] M. Strübe. *Packet reception traces from the WSN-Testbed at the I4, Friedrich-Alexander University Erlangen-Nuremberg [Data set]*. 2017. DOI: 10.5281/zenodo.582285 (cit. on pp. 102 sq., 105 sq.).

[Str17b] M. Strübe. *RSSI Traces from the WSN-Testbed at the I4, Friedrich-Alexander University Erlangen-Nuremberg*. 2017. DOI: 10.5281/zenodo.582278 (cit. on pp. 66 sqq., 70 sq.).

[Sub+14] K. S. Subramani, J. D. Beshay, N. Mahabaleshwar, E. Nourbakhsh, B. McMillin, B. Banerjee, R. Prakash, Y. Du, P. Huang, T. Xi, Y. You, J. D. Camp, P. Gui, D. Rajan, and J. Chen. "Wireless Networking Testbed and Emulator (WiNeTestEr)". In: *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2014)*. ACM, 2014, pp. 51–58. ISBN: 978-1-4503-3030-5. DOI: 10.1145/2641798.2641809. (Cit. on pp. 18, 73).

[Sur+13] T. Surmacz, M. Słabicki, B. Wojciechowski, and M. Nikodem. "Lessons Learned from the Deployment of Wireless Sensor Networks". In: *Computer Networks*. Communications in Computer and Information Science 370. Springer Berlin Heidelberg, 2013, pp. 76–85. ISBN: 978-3-642-38864-4, 978-3-642-38865-1. DOI: 10.1007/978-3-642-38865-1_9. (Cit. on p. 53).

[Sze+04] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. "Lessons from a Sensor Network Expedition". In: *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN 2004)*. LNCS 2920. Springer, 2004, pp. 307–322. ISBN: 978-3-540-20825-9, 978-3-540-24606-0. DOI: 10.1007/978-3-540-24606-0_21. (Cit. on pp. 2, 56, 59).

[Tan11] O. Tange. "GNU Parallel - The Command-Line Power Tool". In: *;login: The USENIX Magazine* 1 (2011), pp. 42–47. (Cit. on p. 94).

[Tar13] R. Tartler. "Mastering Variability Challenges in Linux and Related Highly-Configurable System Software". PhD thesis. Friedrich-Alexander-Universit{ä}t Erlangen-N{ü}rnberg, 2013. URL: http://opus4.kobv.de/opus4-fau/files/3613/ReinhardTartlerDissertation.pdf (cit. on p. 45).

[Tex] Texas Instruments. *FRAM Microcontroller - What is FRAM - MSP430 FRAM Series - TI.com*. URL: http://www.ti.com/lsds/ti/microcontrollers%7B%5C_%7D16-bit%7B%5C_%7D32-bit/msp/ultra-low%7B%5C_%7Dpower/msp430frxx%7B%5C_%7Dfram/what%7B%5C_%7Dis%7B%5C_%7Dfram.page?DCMP=ep-mcu-msp-fram-en%7B%5C&%7DHQS=ep-mcu-msp-fram-b-lp-en%7B%5C#%7Dfram-base (visited on 09/17/2015) (cit. on p. 14).

[Tex04] Texas Instruments. *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Applications*. 2004 (cit. on pp. 13, 29 sq., 36, 56, 61, 75 sq.).

[Tex07] Texas Instruments. *CC1000 Single Chip Very Low Power RF Transceiver - Datasheet*. 2007 (cit. on pp. 27 sq.).

[Tex13] Texas Instruments. *CC1101 Low-Power Sub-1 GHz RF Transceiver*. 2013 (cit. on pp. 13, 29).

[Tsi+09] N. Tsiftes, A. Dunkels, Zhitao He, T. Voigt, Z. He, and T. Voigt. "Enabling Large-Scale Storage in Sensor Networks with the Coffee File System". In: *Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN 2009)*. IEEE Computer Society, 2009, pp. 349–360. ISBN: 978-1-4244-5108-1. (Cit. on p. 14).

[UC10] M. Uddin and C. Castelluccia. "Toward Clock Skew Based Wireless Sensor Node Services". In: *Proceedings of the 5th Annual ICST Wireless Internet Conference (WICON 2010)*. 2010, pp. 1–9 (cit. on p. 59).

[Ulb14] P. Ulbrich. "Ganzheitliche Fehlertoleranz in eingebetteten Softwaresystemen". PhD thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2014. URL: http://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/5056 (cit. on p. 41).

[Vig00] J. R. Vig. "Quartz Crystal Resonators and Oscillators". PhD thesis. Fort Monmouth, NJ 07702, USA, 2000. URL: http://www.ieee-uffc.org/frequency-control/learning-vig-tut.asp (cit. on p. 58).

[Vil+11] F. J. Villanueva, M. Daum, M. Strube, J. C. Lopez, R. Kapitza, and F. Dressler. "Deployment-Aware Energy Model for Operator Placement in Sensor Networks". In: *Proceedings of the 11th International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS 2011)*. Department of Information Technologies and Systems, University of Castilla-La Mancha, Spain. IEEE, 2011, pp. 1–6. ISBN: 978-1-4577-0512-0. DOI: 10.1109/DCOSS.2011.5982211. (Cit. on p. 40).

[Wan+14] J. Wang, Y. Liu, Y. He, W. Dong, and M. Li. "QoF: Towards Comprehensive Path Quality Measurement in Wireless Sensor Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 4 (2014), pp. 1003–1013. ISSN: 1045-9219. DOI: 10.1109/TPDS.2013.98. (Cit. on p. 44).

[WAP01] B. Warneke, B. Atwood, and K. S. Pister. "Smart Dust Mote Forerunners". In: *Proceedings of the 14th IEEE International Conference on Micro Electro Mechanical Systems (MEMS 2001)*. 2001, pp. 357–360. DOI: 10.1109/MEMSYS.2001.906552 (cit. on p. 2).

[War+02] B. . Warneke, M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. Pister. "An Autonomous 16 mm^3 Solar-Powered Mode for Distributed Wireless Sensor Networks". In: *Proceedings of IEEE Sensors*. IEEE, 2002, pp. 1510–1515. ISBN: 0-7803-7454-1. DOI: 10.1109/ICSENS.2002.1037346. (Cit. on pp. 2 sq.).

[Whi+05] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. "Exploiting the Capture Effect for Collision Detection and Recovery". In: *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets 2005)*. IEEE Computer Society, 2005, pp. 45–52. ISBN: 0-7803-9246-9. (Cit. on p. 36).

[Wik15a] Wikipedia. *Algorithms for calculating variance — Wikipedia, The Free Encyclopedia*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Algorithms%7B%5C_%7Dfor%7B%5C_%7Dcalculating%7B%5C_%7Dvariance%7B%5C&%7Doldid=682781972 (cit. on p. 50).

[Wik15b] Wikipedia. *Stuxnet — Wikipedia, The Free Encyclopedia*. 2015. URL: http://en.wikipedia.org/w/index.php?title=Stuxnet%7B%5C&%7Doldid=663108587 (cit. on p. 10).

[Wik15c] Wikipedia Contributors. *Free-space path loss*. 2015. URL: http://en.wikipedia.org/w/index.php?title=Free-space%7B%5C_%7Dpath%7B%5C_%7Dloss%7B%5C&%7Doldid=651671832 (visited on 03/10/2015) (cit. on p. 32).

[Wik16] Wikipedia. *Simulink — Wikipedia{,} The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Simulink%7B%5C&%7Doldid=731009775 (cit. on p. 22).

[Wu+12] K. Wu, H. Tan, H.-L. Ngan, Y. Liu, and L. M. Ni. "Chip Error Pattern Analysis in IEEE 802.15.4". In: *IEEE Transactions on Mobile Computing* 4 (2012), pp. 543–552. ISSN: 1536-1233. DOI: 10.1109/TMC.2011.44. (Cit. on p. 28).

[Yan+12] Z. Yang, L. Cai, Y. Liu, and J. Pan. "Environment-Aware Clock Skew Estimation and Synchronization for Wireless Sensor Networks". In: *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM 2012)*. IEEE, 2012, pp. 1017–1025. ISBN: 978-1-4673-0775-8. DOI: 10.1109/INFCOM.2012.6195457. (Cit. on p. 59).

[YK14] K. S. Yildirim and A. Kantarci. "Time Synchronization Based on Slow-Flooding in Wireless Sensor Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 1 (2014), pp. 244–253. ISSN: 1045-9219. DOI: 10.1109/TPDS.2013.40. (Cit. on p. 54).

[YNB11] D. Yu, P. Nanda, and R. Braun. "Credibility Problems and Tradeoff between Realistic and Abstraction in WANET and WSN Simulation". In: *Proceedings of the 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2011)*. 2011, pp. 1–4. DOI: 10.1109/wicom.2011.6040378. (Cit. on pp. 58, 61).

[Yoo+07] S.-e. Yoo, J.-e. Kim, T. Kim, S. Ahn, J. Sung, and D. Kim. "A2S: Automated Agriculture System based on WSN". In: *Proceedings of the IEEE International Symposium on Consumer Electronics 2007*. IEEE, 2007, pp. 1–5. ISBN: 978-1-4244-1109-2. DOI: 10.1109/ISCE.2007.4382216. (Cit. on p. 53).

[Zac+12] S. Zacharias, T. Newe, S. O'Keeffe, and E. Lewis. "Identifying Sources of Interference in RSSI Traces of a Single IEEE 802.15.4 Channel". In: *Proceedings of the 8th International Conference on Wireless and Mobile Communications (ICWMC 2012)*. 2012, pp. 408–414. ISBN: 978-1-61208-203-5. (Cit. on p. 29).

[ZG03] J. Zhao and R. Govindan. "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks". In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, 2003, pp. 1–13. ISBN: 1581137079. DOI: 10.1145/958491.958493. (Cit. on pp. 16, 36).

[Zha+04] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. "Hardware Design Experiences in ZebraNet". In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*. ACM Press, 2004, p. 227. ISBN: 1581138792. DOI: 10.1145/1031495.1031522. (Cit. on p. 10).

[Zha+07] L. Q. Zhang, F. Wang, M. K. Han, and R. Mahajan. "A General Model of Wireless Interference". In: *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom 2007)*. ACM Press, 2007, p. 171. ISBN: 9781595936813. DOI: 10.1145/1287853.1287874. (Cit. on p. 36).

[Zho+04] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. "Impact of Radio Irregularity on Wireless Sensor Networks". In: *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSYS 2004)*. ACM Press, 2004, p. 125. ISBN: 1581137931. DOI: 10.1145/990064.990081. (Cit. on pp. 34, 61).

[ZHS12] Z. Zhao, W. Huangfu, and L. Sun. "NSSN: A Network Monitoring and Packet Sniffing Tool for Wireless Sensor Networks". In: *Proceedings of the 8th International Wireless Communications and Mobile Computing Conference (IWCMC 2012)*. IEEE, 2012, pp. 537–542. ISBN: 978-1-4577-1379-8. DOI: 10.1109/IWCMC.2012.6314261. (Cit. on p. 54).

[Zim+12] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. "pTunes: Runtime Parameter Adaptation for Low-Power MAC Protocols". In: *Proceedings of the 11th International Conference on Information Processing in Sensor Networks (IPSN 2012)*. ACM, 2012, pp. 173–184. ISBN: 978-1-4503-1227-1. DOI: 10.1145/2185677.2185730. (Cit. on pp. 45, 73).

[ZK04] M. Zuniga and B. Krishnamachari. "Analyzing the Transitional Region in Low Power Wireless Links". In: *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004)*. IEEE, 2004, pp. 517–526. ISBN: 0-7803-8796-1. DOI: 10.1109/SAHCN.2004.1381954. (Cit. on pp. 16, 35 sqq.).

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Network

ADC Analogue Digital Converter

API Application Programming Interface

BAN Body Area Network

BER Bit Error Rate

BLE Bluetooth Low Energy

BT Bluetooth

CCA Clear Channel Assessment

CCR Channel Check Rate

CPP C PreProcessor

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CSMA Carrier Sense Multiple Access

CSMA-CA CSMA with Collision Avoidance

CSMA-CD CSMA with Collision Detection

DGRM Directed Graph Radio Medium

DSSS Direct-Sequence Spread Spectrum

EEPROM Electrically Erasable Programmable Read-Only Memory

FSK Frequency Shift Keying

GPS Global Positioning System

GUI Graphical User Interface

HAL Hardware Abstraction Layer

HF High Frequency

HVAC Heating, Ventilating, and Air Conditioning

I^2C Inter-Integrated Circuit

ILS Instruction Level Simulator

I/O Input / Output

IoT Internet of Things

IP Internet Protocol

IPv4 IP version 4

IPv6 IP version 6

ISM Industrial, Scientific and Medical

ISR Interrupt Service Routine

JNI Java Native Interface

JTAG Joint Test Action Group

JVM Java Virtual Machine

LAN Local Area Network

LED Light Emitting Diode

LQI Link Quality Indication

MAC Medium Access Control

MANET Mobile Ad hoc NETwork

NCFSK NonCoherent FSK

NFC Near Field Communication

NLS Network Level Simulator

NRZ Non-Return-to-Zero

NV-RAM Non-Volatile Random-Access Memory

OQPSK Offset Quadrature PSK

OS Operating System

OTA Over The Air

PC Program Counter

PCB Printed Circuit Board

pdf Probability Density Function

PN Pseudo-random Noise

PRNG Pseudo Random Number Generator

PRR Packet Reception Ratio

PSK Phase Shift Keying

PWM Pulse Width Modulation

QoS Quality of Service

QPSK Quadrature PSK

RAM Random Access Memory

RCS Revision Control System

RDC Radio Duty Cycling

RF Radio Frequency

RISC Reduced Instruction Set Computer

RNG Random Number Generator

RSSI Receive Signal Strength Indicator

SINR Signal-to-Interference-Plus-Noise Ratio

SLS System Level Simulator

SNR Signal-to-Noise Ratio

SOC System On a Chip

SPI Serial Peripheral Interface

SRAM Static Random-Access Memory

TCP Transmission Control Protocol

TDMA Time Division Multiple Access

TI Texas Instruments

TOF Time of Flight

UART Universal Asynchronous Receiver Transmitter

UAV Unmanned Aerial Vehicle

UDG Unit Disk Graph

UDGM Unit Disk Graph Model

UDP User Datagram Protocol

WLAN Wireless LAN

WSN Wireless Sensor Network

Matt W. said that WSN simulators are not good enough for my approach.
It seems like I was able prove him wrong.