Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Design of a Self-Organizing MAC Protocol for Dynamic Multi-Hop Topologies

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

vorgelegt von

Clemens Mühlberger

aus

Rosenheim

Fakultät für Mathematik und Informatik
Lehrstuhl für Informatik V

Würzburg 2018

Design of a Self-Organizing MAC Protocol for Dynamic Multi-Hop Topologies

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

vorgelegt von

Clemens Mühlberger

aus
Rosenheim

Würzburg 2018

Eingereicht am: 09.08.2017
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Reiner Kolla
2. Gutachter: Prof. Dr. Jörg Nolte

Tag der mündlichen Prüfung: 20.02.2018

Acknowledgments

It always seems impossible until it's done.

Nelson Mandela

This work is the result of my research studies on the field of Wireless Sensor Networks. These studies have been carried out at the Institute of Computer Science at the Julius-Maximilians-University of Würzburg. It was a great pleasure for me to having had the opportunity to work and research at the Chair for Computer Engineering V as research assistant and Ph.D. candidate.

Therefore, I would like to thank my Ph.D. adviser Prof. Dr. Reiner Kolla for the facilitation of my research, especially his support and patience with me.

Next, I would like to thank my second reviewer Prof. Dr. Jörg Nolte for his time and efforts in evaluating my work.

Furthermore, I would also like to thank my proof-reader and long-time companion Prof. Dr. Marcel Baunach, for his constructive spirit, collaborative work and common time.

I would also like to thank my research colleagues and forerunners (in alphabetical order): Dr. Christian Appold, Prof. Dr. Marcel Baunach, Dr. Jürgen Bregenzer, and Dr. Armin Runge for the meaningful (sometimes meaningless but amusing) discussions and for sharing ideas.

Finally, I wish to say a special thank you to my beloved family and, in particular, to my wife Filiz Mühlberger – for unlimited support, invaluable personal advice, and your everlasting faith in me. Especially, during the exceptionally difficult times in 2016 you always offered your strong shoulder to lean on.

Abstract

Biologically inspired self-organization methods can help to manage the access control to the shared communication medium of Wireless Sensor Networks. One lightweight approach is the primitive of desynchronization, which relies on the periodic transmission of short control messages – similar to the periodical pulses of oscillators. This primitive of desynchronization has already been successfully implemented as MAC protocol for single-hop topologies. Moreover, there are also some concepts of such a protocol for multi-hop topologies available. However, the existing implementations may handle just a certain class of multi-hop topologies or are not robust against topology dynamics. In addition to the sophisticated access control of the sensor nodes of a Wireless Sensor Network in arbitrary multi-hop topologies, the communication protocol has to be lightweight, applicable, and scalable. These characteristics are of particular interest for distributed and randomly deployed networks (e.g., by dropping nodes off an airplane).

In this work we present the development of a self-organizing MAC protocol for dynamic multi-hop topologies. This implies the evaluation of related work, the conception of our new communication protocol based on the primitive of desynchronization as well as its implementation for sensor nodes. As a matter of course, we also analyze our realization with regard to our specific requirements. This analysis is based on several (simulative as well as real-world) scenarios. Since we are mainly interested in the convergence behavior of our protocol, we do not focus on the "classical" network issues, like routing behavior or data rate, within this work. Nevertheless, for this purpose we make use of several real-world testbeds, but also of our self-developed simulation framework.

According to the results of our evaluation phase, our self-organizing MAC protocol for WSNs, which is based on the primitive of desynchronization, meets all our demands. In fact, our communication protocol operates in arbitrary multi-hop topologies and copes well with topology dynamics. In this regard, our protocol is the first and only MAC protocol to the best of our knowledge. Moreover, due to its periodic transmission scheme, it may be an appropriate starting base for additional network services, like time synchronization or routing.

Kurzfassung

Biologisch inspirierte, selbst-organisierende Methoden können dabei helfen, die Zugriffskontrolle drahtloser Sensornetze auf das gemeinsame Kommunikationsmedium zu regeln. Ein leichtgewichtiger Ansatz ist das Primitiv der Desynchronisation, das auf einer periodischen Übertragung kurzer Kontrollnachrichten beruht – ähnlich den periodischen Impulsen eines Oszillators. Dieses Primitiv der Desynchronisation wurde bereits erfolgreich als MAC-Protokoll für Single-Hop Topologien implementiert. Außerdem existieren auch einige Multi-Hop Konzepte dieser Protokolle. Allerdings können die verfügbaren Implementierungen nur eine bestimmte Klasse von Multi-Hop Topologien bedienen oder sie sind nicht robust genug gegenüber Veränderungen der Netzwerktopologie. Zusätzlich zu dieser ausgeklügelten Zugriffskontrolle der Sensorknoten eines drahtlosen Sensornetzes in beliebigen Multi-Hop Topologien muss das Kommunikationsprotokoll leichtgewichtig, effizient anwendbar und skalierbar sein. Diese Eigenschaften sind insbesondere für verteilte und zufällig (z.B. durch den Abwurf von Sensorknoten aus einem Flugzeug) aufgebaute Netzwerke von Interesse.

In dieser Arbeit präsentieren wir die Entwicklung eines selbst-organisierenden MAC Protokolls für dynamische Multi-Hop Topologien. Dies beinhaltet die Auswertung damit verbundener Arbeiten, der Konzeption unseres neuen, auf dem Primitiv der Desynchronisation basierenden Kommunikationsprotokolls sowie dessen Umsetzung für Sensorknoten. Selbstverständlich untersuchen wir unsere Realisierung hinsichtlich unserer spezifischen Anforderungen. Diese Analyse basiert auf verschiedenen (simulativen, wie auch aus echter Hardware bestehenden) Szenarien. Da wir vornehmlich am Konvergenzverhalten unseres Protokolls interessiert sind, legen wir unser Augenmerk in dieser Arbeit nicht auf die „klassischen" Netzwerkthemen, wie Routing-Verhalten oder Datenrate. Nichtsdestotrotz nutzen wir hierfür verschiedene realitätsnahe Testumgebungen, aber auch unsere selbstentwickelte Simulationsumgebung.

Gemäß den Ergebnissen unserer Evaluationsphase erfüllt unser auf dem Primitiv der Desynchronisation basierendes, selbst-organisierendes MAC Protokoll für drahtlose Sensornetze all unsere Anforderungen. Tatsächlich funktioniert unser Kommunikationsprotokoll in beliebigen Multi-Hop Topologien und kann zudem gut mit Veränderungen der Topologie umgehen. In dieser Hinsicht ist – nach unserem besten Wissen – unser Protokoll das erste und einzige MAC Protokoll. Außerdem bietet sich unser Kommunikationsprotokoll aufgrund seines periodischen Übertragungsschemas als geeigneter Ausgangspunkt für weitere Netzwerkdienste, wie Zeitsynchronisation oder Routing, an.

Contents

I Introduction 1

1 Introduction 3
1.1 Motivation . . . 3
1.2 Requirements . . . 4
1.3 Scientific Contribution . . . 5

2 Basics 7
2.1 Definitions . . . 7
2.2 Analysis Techniques . . . 22
2.2.1 Real-World Testbed . . . 23
2.2.2 Simulation . . . 24
2.2.3 Summary . . . 25
2.3 Simulation Framework . . . 25
2.3.1 Motivation . . . 25
2.3.2 Simulator extDeSIMc . . . 27
2.3.3 Simulation Model . . . 32
2.3.4 Generator Script . . . 32
2.4 Sensor Node Framework . . . 34
2.4.1 SNoW^5 Sensor Node . . . 34
2.4.2 eZ430 Chronos . . . 36
2.4.3 SuperG Gateway Node . . . 37
2.4.4 SmartOS . . . 38
2.4.5 Sniffer . . . 39

II Desynchronization 41

3 Desynchronization 43
3.1 Introduction . . . 43
3.2 Pulse-Coupled Oscillator Framework . . . 44
3.3 Using PCOs to Synchronize WSNs . . . 45
3.3.1 Adaptation . . . 46
3.3.2 Related Work . . . 47
3.4 Using PCOs to Desynchronize WSNs . . . 47

4 Desynchronization as MAC Protocol 49
4.1 Generic Framework . . . 49

4.2 General Conclusions . . . 53
4.2.1 Single-Hop Topology . . . 53
4.2.2 Multi-Hop Topology . . . 55
4.3 The Midpoint Approach . . . 59
4.3.1 Proof of Convergence . . . 61
4.3.2 Related Work . . . 67
4.4 The Local Max Degree Approach . . . 68
4.4.1 Related Work . . . 69
4.5 The Frog-Call Inspired Approach . . . 70
4.5.1 Related Work . . . 71
4.6 The Artificial Force Field Approach . . . 71
4.6.1 Related Work . . . 73

5 The extended-Desync Protocol 75
5.1 Motivation . . . 75
5.2 The Hidden Terminal Problem Revised . . . 76
5.2.1 The Local Max Degree . . . 77
5.2.2 The RTS/CTS Handshake . . . 77
5.3 Phase Shift Propagation . . . 79
5.3.1 Basic Idea . . . 79
5.3.2 Constraint Graph Creation . . . 80
5.3.3 Timed Constraint Graph . . . 81
5.4 Timing Issues . . . 81
5.4.1 Communication Delays . . . 82
5.4.2 Timestamping . . . 85
5.5 Neighbor Information . . . 87
5.5.1 Sample Scenario . . . 88
5.5.2 Naïve Approach . . . 88
5.5.3 Phase Shift Approach . . . 90
5.5.4 Reciprocal Phase Shift Approach . . . 92
5.5.5 Summary . . . 94
5.6 Information Packing . . . 95
5.6.1 SmartNET Support . . . 95
5.6.2 Naïve Approach . . . 98
5.6.3 Fixed Size Subset Approach . . . 98
5.6.4 Fixed Percentage Approach . . . 99
5.6.5 Summary . . . 100
5.7 Frame Structure . . . 101
5.8 Practical Issues . . . 102
5.8.1 Nodes In . . . 102
5.8.2 Nodes Out . . . 104
5.9 Summary . . . 105

6 The extended-Desync$^+$ Protocol 109
6.1 Motivation . . . 109

6.2 Stale Information Problem . . . 110
6.2.1 Single-Hop Topology . . . 110
6.2.2 Multi-Hop Topology . . . 112
6.3 Refractory Threshold . . . 113
6.3.1 Basic Idea . . . 114
6.3.2 Impact . . . 115
6.3.3 Related Work . . . 116
6.4 Excursion: Pseudo Random Number Generator (PRNG) . . . 117
6.4.1 PRNG Implementation for our Sensor Node Framework . . . 117
6.4.2 PRNG Implementation for our Simulation Framework . . . 118
6.5 Summary . . . 119

III Evaluation 121

7 Analysis 123
7.1 Simulation Model Validation . . . 123
7.1.1 Object of Investigation . . . 123
7.1.2 Expectation . . . 125
7.1.3 Procedure . . . 126
7.1.4 Results . . . 126
7.2 Setup Consequences . . . 130
7.2.1 Object of Investigation . . . 131
7.2.2 Expectation . . . 131
7.2.3 Procedure . . . 133
7.2.4 Results . . . 134
7.3 Jump Size Parameter . . . 136
7.3.1 Object of Investigation . . . 136
7.3.2 Expectation . . . 136
7.3.3 Procedure . . . 137
7.3.4 Results . . . 138
7.4 Refractory Threshold . . . 139
7.4.1 Object of Investigation . . . 141
7.4.2 Expectation . . . 141
7.4.3 Procedure . . . 141
7.4.4 Results . . . 142
7.5 Applicability . . . 145
7.5.1 Object of Investigation . . . 145
7.5.2 Expectation . . . 146
7.5.3 Procedure . . . 146
7.5.4 Results . . . 146
7.6 Summary . . . 149

8 Discussion **151**
8.1 Outlook . . . 151
8.1.1 Adaptive Jump Size Parameter . . . 151
8.1.2 Adaptive Refractory Threshold . . . 152
8.1.3 Additional Objects of Investigation . . . 153
8.2 Add-Ons . . . 154
8.2.1 Time Synchronization . . . 154
8.2.2 Routing . . . 155
8.2.3 Distributed Data Management . . . 156

9 Summary and Conclusion **159**
9.1 Summary . . . 159
9.2 Conclusion . . . 160

IV Lists and Indexes **161**

Bibliography **163**

List of Figures **181**

List of Tables **183**

List of Listings **185**

Acronyms **187**

Part I

Introduction

Es ist nicht genug, zu wissen,
man muß auch anwenden;

Johann Wolfgang von Goethe

Abstract

This part motivates for researching on the conceptional design and analysis of a self-organizing communication protocol for arbitrary topologies in Wireless Sensor Networks based on the primitive of desynchronization. In this regard, this part not only introduces this certain topic, but also specifies the underlying terms and tools of this work.

In particular, Chapter 1 gives reasons for the development of a self-organizing communication protocol for WSNs. Referring to this, the necessary requirements, which have to be fulfilled by our communication protocol, are also specified in this chapter. Chapter 2 defines the essential terms and presents the particular tools, which are utilized for the purpose of this work. It also discusses our research approach and outlines the applied methodology. Furthermore, this chapter introduces the hardware as well as the software framework used for development and evaluation of our communication protocol.

Chapter 1

Introduction

1.1 Motivation

The evolution in the areas of semiconductors and computers over the last decades allows for the tight integration of complex sensors and actuators as well as computational units. This development results not only in smaller and more powerful computing platforms but also in more energy-efficient and globally connected devices. This is one of the keys for the production of small network components which are able to communicate wirelessly to achieve (common) goals and to provide distributed services.

The availability of these network components established new fields of research and development. Different aspects may be in focus, depending on the intended application domain. Along with the technology, there was also an evolution in the naming of such networks, beginning with Wireless Sensor Network (WSN)[1] to Cyber Physical System (CPS)[2] to Internet of Things (IoT)[3] – to name but a few. Nevertheless, the fundamental problems and central questions, e.g., regarding the dependability of the underlying wireless communication, remained the same or similar. For instance, the European Workshop for Wireless Sensor Networks (EWSN) did change its name to *International Conference on Embedded Wireless Systems and Networks* (EWSN) as it did also expand its scope from Wireless Sensor Network to the fields of Cyber Physical System and Internet of Things in addition. A clear distinction between these particular network types seems to be not always obvious (cf. [117]). Hence, even though we will mainly use the term Wireless Sensor Network within this work, our propositions and statements are also valid for other tpyes of Wireless Networks (WNs) in principle.

Saying this, all Wireless Networks, and Wireless Sensor Networks in particular, do require a well-defined communication stack to successfully exchange data, and thus, to perform certain tasks in the desired (and efficient) way. For instance, a large variety of communication protocols for WSNs already does exist. However, the applicability of some of these protocols is restricted to a certain network size and/or to a specific network structure. Apart, highly dynamic networks, e.g., systems with mobile devices, may not rely on a fixed infrastructure nor on a central control unit. Therefore, we will aim for a communication protocol which scales well and is applicable in arbitrary topologies. In this regard, we present a decentralized and distributed approach to establish a flexible and adaptive communication infrastructure for a dependable data exchange within such networks.

[1] For instance, the *ACM Conference on Embedded Networked Sensor Systems* (Sensys) was established in 2003, the *European Workshop for Wireless Sensor Networks* (EWSN) was established in 2004, and the *Fachgespräch Sensornetze* (FGSN) was established in 2003

[2] For instance, the *CPS Week* was established in 2008 and the *ACM/IEEE International Conference on Cyber-Physical Systems* (ICCPS) was established in 2010.

[3] For instance, the *Internet of Things Conference* was established in 2008.

In order to accomplish related communication challenges, Nature often demonstrates an impressive diversity regarding efficient ways of coping with biological or environmental limitations and the laws of nature. Over hundreds and thousands of years several approaches have evolved and were tested and refined or discarded by different creatures and organisms to develop impressive concepts. For instance, cardiac cells show coordinated beating or fireflies flash in unison. This spontaneous emergence of collective synchronization does not rely on a central control unit. Instead, the components just act on perception and simple rules. This *primitive of desynchronization* is the basis for this kind of self-organizing communication. We want to adapt this biological concept for our communication protocol.

To sum up, the objective of this work is the

- *conceptional design and analysis* of a
- *self-organizing*
- *communication protocol* for
- *arbitrary topologies* in
- *Wireless Sensor Networks* based on the
- *primitive of desynchronization*.

Our particular requirements for such a communication protocol will be specified in the next section.

1.2 Requirements

Having drafted the objective of this work, i.e., the development of a self-organizing communication protocol for arbitrary topologies in WSNs, we still need to specify in detail the scope of this protocol. In particular, such a protocol has to meet our demands as follows:

1. First of all, we focus on the *Medium Access Control* (MAC) mechanism of the network components as main service of our wireless communication protocol. Designed for tiny embedded systems, this Medium Access Control protocol has to be lightweight and applicable. This means that the protocol introduces just a small overhead and thus can be executed by network components of potentially low computational power and constrained resources, like little memory.

2. Next, wireless communication in a network always involves the risk of packet collisions and data loss. In general, these packet collisions are undesired as they might lead to violated real-time constraints, destroy information, and cause additional expenses in time and energy, e.g., for retransmissions. Therefore, we aim on minimizing the probability of such packet collisions by developing a *Time-Division Multiple Access* (TDMA) scheme with exclusively assigned transmission slots.

3. In this regard, we want to prevent the single point of failure of centralized synchronization approaches as well as the rigidity of an a priori schedule. Hence, we decided to rely on neither a global clock nor an explicit time synchronization protocol. In contrast, the system should be robust against topology dynamics, e.g., joining or leaving network components. Therefore, we have developed a self-organizing MAC protocol based on a dynamic TDMA schedule that is continuously and individually maintained by each network node. The underlying self-organizing algorithm is mainly influenced by the biologically inspired primitive of desynchronization. Consequently, the probability of collisions will be minimized – still enabling the system to adapt rapidly to topology dynamics.

4. Indeed, there already exist some MAC protocols that are based on this primitive of desynchronization. However, most of these protocols are applicable for single-hop WSNs only. In contrast, our protocol shall be generally applicable, i.e., it has to operate well in arbitrary network types, including multi-hop topologies. Additionally, the self-organizing protocol has to scale well with the network size, i.e., its performance has to be independent of the number of network components and links.

5. Finally, due to significant mobility or environmental perturbation in WSNs, a fast convergence of the schedule adaptation is also required in case of such topology changes. While nodes might operate on different schedules during this short time of disorder, there should be no packet collisions and no data loss. However, due to the self-organizing manner, there are trade-offs, for instance between convergence rate and protocol overhead.

To sum up, for the communication within Wireless Sensor Networks we are in need of a lightweight and applicable as well as scalable MAC protocol, which implements a self-organizing and dynamic TDMA schedule. This communication protocol not only has to support arbitrary (multi-hop) topologies and has to be robust against environmental perturbations, but also has to converge efficiently to an adapted schedule in case of topology changes. As mentioned before, there already exists a wide variety of communication protocols for Wireless Sensor Networks. Nevertheless, a communication protocol for Wireless Sensor Networks that meets all our demands from above is still missing.

1.3 Scientific Contribution

The focus of this work is on the design of a self-organizing communication protocol for Wireless Sensor Networks. In particular, we have developed and analyzed a self-organizing MAC protocol which is applicable for arbitrary multi-hop topologies being subject to topology dynamics. In this regard, we are mainly interested in the convergence behavior of our protocol. Therefore, we do not focus on the "classical" network issues, like routing behavior, data rate, network throughput, and channel utilization. Thus, this work is organized as follows:

Part I motivates research in this particular topic and also specifies the fundamental terms and definitions which are used within this work. Moreover, we discuss our research approach and outline the applied methodology, whose result is tested through a mix of simulation and experiments. In this regard, we present the framework which was used for development as

well as for evaluation: On the one hand, we deployed real hardware in real-world testbeds, i.e., the network components, which executed embedded software, e.g., the embedded operating system as well as applications. On the other hand, we developed a simulation framework, which allows an efficient prediction and comfortable *ex post facto* analysis of the settling process of Wireless Sensor Networks implementing our self-organizing MAC protocol.

Part II addresses the core concepts of this work. The initial task is to introduce the biologically inspired *primitive of desynchronization*. This primitive is the basis for several approaches to synchronize but also to desynchronize Wireless Sensor Networks. Furthermore, implementations of this primitive of desynchronization as MAC protocol for WSNs already exist. Within this part, these implementations are discussed and opposed to our requirements from Section 1.2. Since none of the available MAC protocols based on the primitive of desynchronization met our demands, we developed a self-organizing MAC protocol with a dynamic TDMA schedule for arbitrary WSN topologies. We also attempt to prove the convergence of our approach for multi-hop topologies under certain assumptions. The development process and our resulting protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, respectively, are described in detail.

The evaluation of our EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocol is presented in Part III. Here, we do confront our protocols with our requirements from Section 1.2. This is done by analyzing and discussing the results of appropriate scenarios – realized as simulations or as real-world testbeds. To the best of our knowledge, our communication protocols EXTENDED-DESYNC as well as EXTENDED-DESYNC$^+$ are the first and only MAC protocols for WSNs, which are based on the primitive of desynchronization, and which are able to operate in arbitrary multi-hop topology with topology dynamics at the same time. Subsequent to this evaluating chapter, we give an outlook to future research directions. Moreover, we present some potential add-ons to our MAC protocol which will offer additional (network) services. Part III closes with a conclusion about this work.

Part IV contains additional information, like lists and references. These lists are provided for a more comprehensive understanding of various details within this work.

To sum up, the environment of Wireless Sensor Networks may stimulate topology dynamics due to node failures or mobility. Nevertheless, the stability and robustness of the communication network is essential to fulfill a common task (e.g., distributed real-time control in wirelessly connected and autonomously driving cars). Especially for arbitrary multi-hop topologies, a lightweight but adaptive communication protocol, which is able to cope with such conditions, is missing. In this regard, we developed a MAC protocol for WSNs, which combines the advantages of a TDMA scheme with the benefits of a self-organizing approach to meet our demands.

Chapter 2

Basics

Abstract

The number of published literature studying the field of Wireless Networks in general, and Wireless Sensor Networks in particular, increased within the past years. As a consequence, the meaning of some terms differs in literature nowadays. Therefore, we first state in Section 2.1 our interpretation of the key terms as used within this work. Likewise, the family of sensor node systems also grew within the past years. Thus, in Section 2.2 we briefly introduce potential analysis techniques to evaluate our approach. Finally, we describe the tools we used for our simulations and experiments, namely our self-developed simulator extDeSIMc in Section 2.3 as well as our sensor node hardware and software in Section 2.4.

2.1 Definitions

As stated in Chapter 1, the main subject of our analyses will be a *self-organizing* radio communication protocol for Wireless Networks.

Definition 2.1 Self-Organization. In accordance with [55], we call the process by which the system behavior at a higher level emerges solely from the interactions of system components at a lower level *self-organization*. Especially, when low-level components have just a local view (i.e., they interact according to locally acquired information) and when there is no low-level component with a global view or global knowledge about the current behavior of the other components. Therefore, one main characteristics of self-organizing systems is the absence of an external or a centralized control.

As suggested by Dressler in [55], (a combination of) the following mechanisms can be used to achieve self-organization:

Feedback A positive feedback would reinforce or activate, whereas a negative feedback would diminish or suppress a specific behavior of system components. Feedback may be provided from external or internal system components. Thus, it is strongly related to the acquiring of information.

Interaction System components may interact with other system components or with the environment. Communication is the most common kind of interaction and a prerequisite for coordination. We further distinguish between direct communication, e.g., transmission of a message, and indirect communication, which requires auxiliary means (like whiteboard, environment, or other system components). Moreover, modifying as well as sensing of environmental parameters is also considered as indirect interaction with the environment.

Probabilistic Techniques Randomized algorithms can be used to introduce contingency and arbitrary behavior into the system. The source of randomness can be a *Random Number Generator* (RNG) or – as within this work – a *Pseudo Random Number Generator* (PRNG). There is a brief excursion on Pseudo Random Number Generator in Section 6.4.

The node's decisions are mainly based on just locally available information. Depending on its degree of non-linearity (e.g., the number of nodes), the behavior of such a self-organizing system even may be as unpredictable as desirable. In addition, even optimum configurations for such systems are subject to change frequently, i.e., a fast convergence behavior is required (cf. Section 1.2). Consequently and in accordance to, e.g., [55], the behavior of a self-organizing system may be predictable only to a certain extent when using probabilistic techniques. For instance, statistical measurements are used in [173] to enhance the convergence rate of the so-called Desync-Ort protocol (cf. Section 4.3.2). Moreover, we utilize the *exponentially weighted moving average* (EWMA) (cf. Observation 4.11) to facilitate the system's convergence behavior.

Notably, in the context of this work we consider the transmission of messages in general and of firing messages (cf. Definition 3.2) in particular, as inter-component interaction but not as feedback. Therefore, a combination of interaction and probabilistic techniques is used within this work for our self-organizing algorithm. The main field of application of our self-organizing approach is a *Wireless Ad hoc Network*.

Definition 2.2 Wireless Ad hoc Network, Node. A *Wireless Ad hoc Network* consists of a set N of participating network elements, so-called *nodes*. All (heterogeneous or homogeneous) nodes of a Wireless Ad hoc Network are able to interact wirelessly *ad hoc* through a shared medium, like air (*over-the-air*) or water (cf. [195, 107, 7, 79]). However, a pre-installed infrastructure supporting network operation is missing for this "ad hoc" network type. This means that the infrastructure has to be built on demand, e.g., in a self-organizing manner by the nodes themselves.

Meanwhile, there exist various different (sub)types of Wireless Ad hoc Networks. The most prominent representative is the Mobile Ad hoc Network (MANET), where every node can be mobile, i.e., the spatial location of each node can be non-stationary. In addition, one subtype of the MANET is the *Wireless Sensor Network* (WSN). This network type will be in the foreground within this work.

Definition 2.3 Wireless Sensor Network. The *Wireless Sensor Network* (WSN) is a subtype of the MANET and consists of just a few or up to thousands of so-called *sensor nodes*. To keep costs low, the network components are usually assembled from cheap and simple components. Due to radio interference, sensor node failure as well as leaving or joining sensor nodes, the (arbitrary) network structure of the WSN is subject to frequent changes[1] (cf. [6]).

A WSN is often densely deployed to achieve some sort of intentional *redundancy* since sensor nodes tend to be error-prone. Besides, some WSN applications require a *sink*, i.e., a node collecting all the data from the other sensor nodes of that network, where the data

[1] This may hold for all types of MANETs.

flow is just one-way towards the sink but not vice versa. In contrast to these applications and to permit a broader area of network types, the existence of a sink is neither necessary nor assumed within this work. Finally, as sensor nodes usually exchange only short (sensor) data, a WSN has just low bandwidth requirements. Another subtype of the MANET is the *Wireless Sensor/Actuator Network* (WSAN), where actuators are explicitly enabled to manipulate the environment.

Definition 2.4 Wireless Sensor/Actuator Network. The *Wireless Sensor/Actuator Network* (WSAN) is a subtype of the MANET, which is related to the Wireless Sensor Networks. It consists of just a few or up to thousands of so-called *sensor/actuator nodes*. In contrast to a Wireless Sensor Network, a Wireless Sensor/Actuator Network has the ability to modify and manipulate its environment using actuators. Depending on the current application, the fast reaction on sensor inputs for a corresponding actuator configuration is mandatory. Therefore, real-time capability in both, communication and data processing, is an important requirement in Wireless Sensor/Actuator Networks.

Another subtype of the MANET as special type of the WSAN is the *Vehicular Ad hoc Network* (VANET), where a mobile node corresponds to a moving vehicle.

Definition 2.5 Vehicular Ad hoc Network. The *Vehicular Ad hoc Network* (VANET) is a subtype of the MANET. It consists of mobile nodes, namely *vehicles* or *cars*, and possibly immobile nodes at the roadside, like road signs, street signs, traffic lights, and guide posts. This differentiation allows the communication solely between mobile nodes, called *Inter-Vehicular Communication* (IVC), *car-to-car* (C2C) communication, or *vehicle-to-vehicle* (V2V) communication, as well as the communication between mobile and immobile nodes, called *car-to-infrastructure* (C2I) communication. A VANET is characterized by the potentially short communication time between interacting nodes, which is due to the mobility at (high) velocities of (some) nodes (cf. [115, 53]).

Definition 2.6 Wireless Sensor Node, Wireless Sensor/Actuator Node. Like any network component (cf. Definition 2.2), a *sensor node* as well as a *sensor/actuator node* has the ability to interact with other elements of the network. Therefore, it consists of a microcontroller as central processing unit and an (integrated) RF unit for wireless communication. Figure 2.1 outlines the scheme of a sensor/actuator node and a sensor node, respectively. In order to limit costs and size, the RF transceiver typically supports just half-duplex (cf. Definition 2.9), and its maximum communication range is from some tens up to few hundreds of meters. However, the central microcontroller is often severely limited in computational power as well as in memory, i.e., there are limits concerning the computation speed, the accuracy of the results, and the acceptable amount of input/output values. Therefore, in order to store undelivered messages, supplementary logged data, miscellaneous configurations, or alternative software images (cf. [13]), an additional non-volatile memory for long-term data storage is often available (cf. Figure 2.1). For programming purposes and enhanced (network) connectivity, a sensor node may support several (wired) interfaces and protocols, like JTAG and RS-232. Unlike the network components from Definition 2.2, sensor/actuator nodes and sensor nodes in particular, are intended to be equipped with sensor technology (cf. Figure 2.1) to measure chemical or physical quantities or changes, like temperature, humidity, or acceleration. Furthermore, a sensor/actuator node in a Wireless Sensor/Actuator Network may

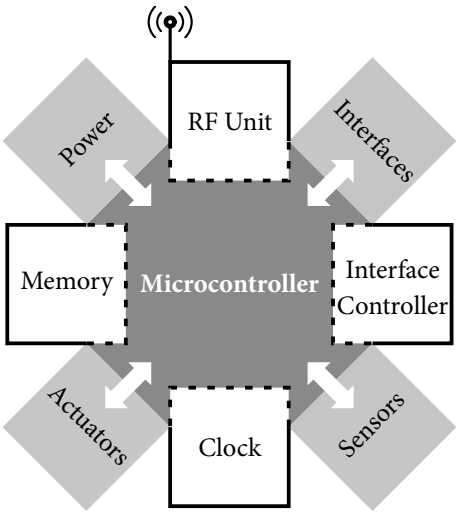


Figure 2.1: The scheme of a sensor/actuator node and – except for actuators – of a sensor node. Ports to the environment are colored in gray. Components in white may be (partially) integrated into the microcontroller.

even be able to control additional actuators, e.g., a display or a stepper motor, and thus also influences and manipulates its surrounding. Mandatory for this work, each sensor/actuator node as well as each sensor node has to provide its own clock to record its local time as well as the timestamps of internal and external events (cf. Figure 2.1).

The spectrum of radio devices used at such wireless platforms ranges from sub 1-GHz transceivers like RFM TR1000/TR1001 (e.g., at early platforms like EYES [188] and ScatterWeb ESB [67]) and Chipcon CC1000/CC1100/CC1101 (e.g., at SNOW^5 [20, 23], BTnode [28], and MICA2 [43]), to 2.4 GHz IEEE 802.15.4/ZigBee compliant transceivers Chipcon CC2420/CC2520 (e.g., at MICAz [46], Telos [144], TelosB/TMote Sky [44], and Imote2 [45]) and System-on-Chip radio transceiver Nordic nRF24 (e.g., at EcoSpire [37]). Or the particular radio device is moved to a separate communication module, as for instance at the WaspMote family [106]. This very variety of radio interfaces as well as the offered communication protocols complicates the interconnection of miscellaneous WSNs. In this regard, more powerful sensor nodes with special capabilities, i.e., to operate as multi-layer gateway or as protocol converter, already exist. Examples include the s-net gateway [66], Meshlium IoT gateway [106], and the SuperG gateway [129] (cf. Section 2.4.3). Since most components of a sensor node consume little energy, most nodes are powered by (rechargeable) battery. Certainly, sensor nodes also may perform energy harvesting by means of appropriate hardware, like HelioMote [147], Prometheus [90], or Everlast [162]. The capability of the SNOW^5 sensor node for energy harvesting is analyzed in [60].

In general, sensor nodes are rather cheap, quite simple in construction, and thus available in large quantities. From an economic point of view, these characteristics effectively facilitate a wide range of applications. Especially, if the environment is rough, unfriendly, or even

hostile (cf. [116]). Examples include applications for agricultural and habitat monitoring [85, 110, 183, 27], applications for wildlife observation [91, 11, 76] and underwater surveillance [107, 7] as well as applications being subject to other destructive conditions, like gunfire [163] or disaster [172].

As a consequence, the failure or malfunction of several sensor nodes of a system within such a harsh environment is very likely. The system has to be designed accordingly to accept this risk, e.g., allowing *desired redundancy* of hardware components or implementing a distributed data management (cf. Section 8.2.3). Indeed, this sort of redundancy and flexibility has to be supported by the used communication protocol as well. Especially, self-organizing protocols appear to be predestined to operate well under these conditions. Nevertheless, a robust and self-organizing communication protocol for WSNs in multi-hop topologies with topology dynamics is still missing. Therefore, we focus on WSNs within this work – even though our protocol may perform quite well for other network types due to its intended robustness. Hence, we use the terms *sensor node*, *sensor/actuator node*, and *node* interchangeably from now on. Indeed, the interaction between two nodes is realized as direct communication via a *channel*.

Definition 2.7 Channel, Link. The communication *channel* corresponds to a logical internode connection over a shared medium, e.g., a radio channel with multiple access. Moreover, at a time a channel transfers data from exactly one *sender* (or *transmitter*) to an arbitrary number of *receivers*. The capacity of a channel, i.e., the channel's capability to transmit information per time unit, is limited and measured by the channel's *bandwidth* or the channel's *data rate*. The *point-to-point* (P2P) connection of two nodes via a communication channel is called *link*.

One approach to overcome issues resulting from multiple access on the shared medium to a certain extent is *Carrier Sense* (CS).

Definition 2.8 Carrier Sense. The *Carrier Sense* (CS) approach supports the avoidance of potential packet collisions on a shared medium. For this purpose, a node tries to listen to (*sense*) the shared medium (*carrier*) right before its own transmission in order to verify the absence of other traffic.

Indeed, we further classify any link by its

- *direction*, i.e., as *unidirectional* or as *bidirectional* link, its
- *duplex*, i.e., half-duplex or full-duplex mode, and by its
- *symmetry*, i.e., as *symmetrical* or as *asymmetrical* link.

Definition 2.9 Uni-/Bidirectional, Half duplex/Full duplex, Symmetrical/Asymmetrical. A link which enables the communication between two nodes in either direction is called *bidirectional*. Whereas, a link between two nodes is called *unidirectional*, if just one of them is able to transmit data to the other one, but never vice versa. A link may be used in *full-duplex mode*, i.e., the radio unit is able to transmit and to receive data at the same time, or a link may be used in *half-duplex mode*, i.e., the radio unit is just able either to transmit or to receive at the same time. At an *asymmetrical* link, the data transmission between both nodes

Preamble	SYNC word	Control Data (Header)	User Data (Payload)	CRC	RSSI	LQI

Figure 2.2: The structure of a potential radio packet. Parts colored in gray are added automatically by the CC1100 radio unit, which is used mostly within our real-world testbeds.

is varying in terms of, e.g., data loss, bandwidth, or data rate. Consequently, a bidirectional link with equal rates is called *symmetrical*.

Observation 2.1. A unidirectional link can be considered as an extreme form of an asymmetrical link.

The communication amongst the nodes within this work is *packet*-based, i.e., a link within this work is able to deliver *packets* only.

Definition 2.10 Packet. A *packet* is a structured unit of data. Within this work, each packet consists of *control data* as well as *user data* (or *payload*) as illustrated in Figure 2.2. The control data provides information which is required by the used protocol(s) for a correct packet handling, e.g., the packet type, the destination address, or a sequence number. This control data typically resides in the *header* or the *trailer* of a packet. The content of the user data depends on the particular application. In contrast to the header, the payload may even be omitted, e.g., for *acknowledgment* (ACK) messages. Header and payload are embedded into additional parts which are required for a successful transmission, e.g., the *preamble*, the *SYNC word* (SYNC), or parts, which specify the current transmission, e.g., the *Cyclic Redundancy Check* (CRC), the *Received Signal Strength Indicator* (RSSI), or the *Link Quality Indicator* (LQI). Depending on the used RF unit, these subsidiary parts may be added by the radio hardware automatically, but are configurable to some extend.

So far, we classified the network types and the network components just technically. Indeed, to analyze the functionality and applicability of our self-organizing communication protocol, a formal description of the *network structure*, also called *topology*, is mandatory.

Definition 2.11 Topology. A *topology* describes the structure of a network, i.e., the links between the nodes of a system. Noteworthy, a topology may change over time (*topology dynamics*) and thus just reflects the present network configuration. Especially, this applies for Mobile Ad hoc Networks with mobile components or Wireless Networks under ever-changing environmental conditions.

The topology can be described formally using concepts from *graph theory* (cf. textbooks on graph theory like [29, 184, 101]):

Definition 2.12 (Network) Graph. A directed *(network) graph* $\vec{G} = (N, E)$ is an ordered pair of disjoint and finite sets N and E. It consists of the set N of vertices, or nodes, and the set $E \subseteq N \times N$ of directed edges, or links. The set of edges in a directed graph is composed of just ordered node pairs: The directed edge $(i, j) \in E$ from a node $i \in N$ to another node $j \in N$ denotes that node j is able to receive signals emitted by node i – but not vice versa, unless there is an edge $(j, i) \in E$. Consequently, the edge $(i, j) \in E$ only exists if node $j \in N$ is

within the communication range of node $i \in N$. Since the wireless communication in WSNs is typically realized as half-duplex (cf. Definition 2.9), self-loops are not contained within a network graph, i.e., for any node $i \in N$ holds $(i,i) \notin E$. Accordingly, a (network) graph $\overline{G}$ is undirected if the set of edges is composed of not ordered node pairs, i.e., for every edge $(i,j) \in E$ holds $(j,i) \in E$.

Since directed (network) graphs can be used to describe more general scenarios than undirected ones, we will focus on the use of directed (netowrk) graphs. Unless otherwise stated, in the following a *(network) graph* G always denotes an *directed (network) graph* $\vec{G}$.

Definition 2.13 Path, (Single) Hop. A *path* from a node $i \in N$ to a node $j \in N$ in the (directed) graph $G = (N,E)$ is a sequence of nodes $(i = i_0, i_1, \ldots, i_{l-1}, i_l = j)$ such that for any $x \in \{0,\ldots,l-1\}$ holds $(i_x, i_{x+1}) \in E$.[2] Each edge in a path is called *hop*. The length of the path is equal to l, i.e., the number of edges in the path. Noteworthy, a path of length 1 consists of just a *single hop*.

To simplify our analysis, we just examine *(weakly) connected* topologies.

Definition 2.14 (Weakly) Connected. Two nodes $i \in N$ and $j \in N$ of an undirected graph $\overline{G}$ are called *connected*, if $\overline{G}$ contains a path from node i to node j. If every pair of nodes in an undirected graph $\overline{G}$ is connected, this graph $\overline{G}$ is said to be *connected*. Within this work, we call a directed graph $\vec{G}$ *weakly connected*, if replacing all of its directed edges with undirected ones results in a connected undirected graph $\overline{G}$.

The functionality of our self-organizing protocol strongly relies on the information a node is able to gather about its *neighbor* nodes.

Definition 2.15 One-Hop Neighbor. A node $j \in N$ in a (directed) graph $G = (N,E)$ is called *one-hop neighbor* of node $i \in N$, if $(j,i) \in E$. That means, node j is a one-hop neighbor of node i. Moreover, the shortest path from node j to node i has length 1.

To describe the integration of a node into the network, we have to specify the *neighborhood* of a node from its point of view.

Definition 2.16 Neighborhood, Degree. In conformity with Definition 2.15, the set $N_1(i) \subseteq N$ of *one-hop neighbors* of a node $i \in N$, i.e., the *one-hop neighborhood* of node i, is

$$N_1(i) = \left\{ j \in N \setminus \{i\} \colon (j,i) \in E \right\}, \tag{2.1}$$

and the set $N_2(i) \subseteq N$ of *two-hop neighbors* of node $i \in N$, i.e., the *two-hop neighborhood* of node i, is

$$N_2(i) = \left\{ k \in N \setminus \{N_1(i) \cup \{i\}\} : \left(\exists j \in N_1(i) \colon (k,j) \in E\right) \right\}. \tag{2.2}$$

Since each node has to receive additional information explicitly due to its local view of the network, we further define the *degree* d_i of a node $i \in N$ as

$$d_i = |N_1(i)|. \tag{2.3}$$

[2] Please note that we exclude self-loops in our (network) graphs according to Definition 2.12

Observation 2.2. As a consequence to Definition 2.16, we neglect outgoing edges and take just incoming links into account for a node's neighborhood. This means that the neighborhood of a node does not consist of nodes, which have been reached ("outbound") but which have been gathered ("inbound").

Observation 2.3. The sets $\{i\}$, $N_1(i)$, and $N_2(i)$ are pairwise disjoint for every $i \in N$.

Network graphs consisting of just a single *connected component* are of special interest:

Definition 2.17 Connected Component. A *connected component* is a maximal connected subgraph of an undirected network graph $\overline{G}$. Each node of $\overline{G}$ belongs to exactly one connected component, as does each edge.

Definition 2.18 Subgraph. A graph $G' = (N', E')$ of a (directed) graph $G = (N, E)$ with $N' \subseteq N$ and $E' \subseteq E$ is called (induced) *subgraph*. It is induced by the subset $N' \subseteq N$ of nodes and contains the edge set $E' = \{(i, j) \in E : i, j \in N'\} \subseteq E$.

Lemma 2.1 Connectivity. *Assuming, the directed network graph $\vec{G}$ is not weakly connected, i.e., the corresponding undirected network graph $\overline{G}$ is not connected. Hence, the network graph $\vec{G}$ (and $\overline{G}$ respectively) has at least two connected components. Each subgraph, represented by a single connected component, can then be considered in isolation as an individual network.*

Proof. According to Definition 2.17, each connected component is a maximal connected subgraph. Since the particular connected components are disjoint, they will never interact with each other. Therefore, each single connected component can be considered in isolation as individual network. □

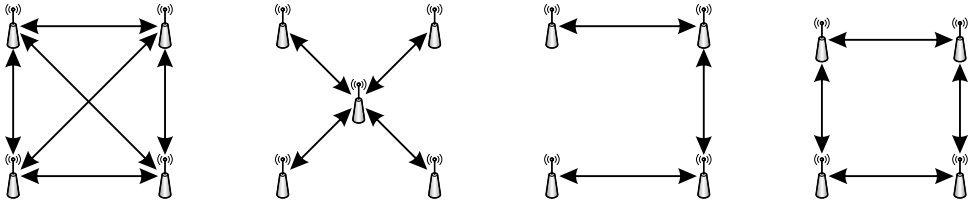
There are some distinct forms of connected components which are of special interest. Due to their plain form, these graphs are easy to understand. Amongst others, these forms are *complete graph*, *star graph*, *line graph*, and *circle graph* (cf. Figure 2.3). Besides, Choochaisri also used these forms to analyze the M-Dwarf protocol in [38].

Definition 2.19 Complete Graph. All nodes in a *complete graph* $C = (N, E)$ are connected to each other, i.e., for any two distinct nodes $i, j \in N$ with $i \neq j$ holds $(i, j) \in E$ and $(j, i) \in E$. Figure 2.3(a) depicts the complete graph C_4 of four nodes.

Definition 2.20 Star Graph. A *star graph* $S = (N, E)$ with $|N| \geq 2$ contains one distinguished node $i \in N$, which is connected to every other node $j \in N$ with $j \neq i$, i.e., for any node $j \in N$ with $i \neq j$ holds $(i, j) \in E$ and $(j, i) \in E$. Apart, there is no connection between any pair of non-distinguished nodes. Figure 2.3(b) depicts the star graph S_5 of five nodes.

Definition 2.21 Line Graph. A *line graph* $L = (N, E)$ with $|N| \geq 2$ is a sequence of nodes with the start node $i_1 \in N$ and the end node $i_{|N|} \in N$, i.e., for $x \in \{1, \ldots, |N| - 1\}$ holds only $(i_x, i_{x+1}) \in E$ and only $(i_{x+1}, i_x) \in E$. Figure 2.3(c) depicts the line graph L_4 of four nodes.

Definition 2.22 Circle Graph. A *circle graph* (or *ring graph*) $R = (N, E)$ with $|N| \geq 2$ is a closed line graph (cf. Definition 2.21), connecting the "start" node $i_1 \in N$ and the "end" node $i_{|N|} \in N$, i.e., for $x \in \{1, \ldots, |N| - 1\}$ holds $(i_x, i_{x+1}) \in E$ and $(i_{x+1}, i_x) \in E$, as well as $(i_{|N|}, i_1) \in E$ and $(i_1, i_{|N|}) \in E$. Figure 2.3(d) depicts the circle graph R_4 of four nodes.



(a) Complete graph C_4 (b) Star graph S_5 (c) Line graph L_4 (d) Circle graph R_4

Figure 2.3: Examples of connected components. An arrow represents the direction of a link from sender to receiver (arrowhead).

Observation 2.4. The definition of a star graph, a line graph, and a circle graph, respectively, sets a minimum number of nodes to distinguish one form from each other. For instance, a star as well as a line with $|N| = 2$ is a complete graph. Moreover, a ring with $|N| = 2$ as well as $|N| = 3$ is also a complete graph. Finally, a star with $|N| = 3$ is indistinguishable from a line with $|N| = 3$.

Within this work, we classify all topologies into *single-hop* and *multi-hop* topologies.

Definition 2.23 Single-Hop Topology. More restrictive than some literature (cf. [93, 200]), but in accordance with the significant related work in Chapter 4, we define a *single-hop topology* as a fully meshed topology, i.e., the corresponding network graph is complete (cf. Definition 2.19). Therefore, the broadcast of any participating node could be received directly by all other nodes of the network (cf. Figure 2.3(a)).

Observation 2.5. For single-hop topologies, $N_2(i) = \varnothing$ always holds.

Definition 2.24 Multi-Hop Topology. Consequently, we define a *multi-hop topology* if the network is not fully meshed, i.e., the corresponding network graph is not complete. Hence, in a multi-hop but (weakly) connected topology exists at least one node $i \in N$ which is not able to interact with every other node $j \in N$ of the network in a direct way, i.e., edge (i, j) or edge (j, i) are missing. Let node j be outside of the communication range of node i, i.e., $(i, j) \notin E$, then node i is called *hidden* from node j.

Observation 2.6. Any topology containing at least one unidirectional link is considered as multi-hop topology.

Observation 2.7. For a node $i \in N$ being hidden from node $j \in N$ always holds $(i, j) \notin E$, and thus $i \notin N_1(j)$.

From Definition 2.24 directly results the *hidden terminal problem*, which was first described by Tobagi and Kleinrock [182].

Definition 2.25 Hidden Terminal Problem. Suppose, a network consists of three nodes a, b, and c, as exemplified in Figure 2.4(a). Nodes a and c can directly transmit to node b, but both nodes a and c are unaware of each other. If at overlapping time intervals node a as well

(a) Hidden terminal problem (b) Exposed terminal problem

Figure 2.4: Sample scenarios of two famous problems being inherent to multi-hop topologies. A gray shape represents the communication range of its centric sender.

as node c transmit a packet to node b, both radio packets collide. As a result, the potential receiver node b receives just corrupt data – if any. Since both nodes a and c are hidden from each other, they cannot overcome this packet collision using Carrier Sense (cf. Definition 2.8) right before their transmissions.

Observation 2.8. Our Definition 2.24 of multi-hop topologies implies that the hidden terminal problem is inherent to multi-hop topologies.

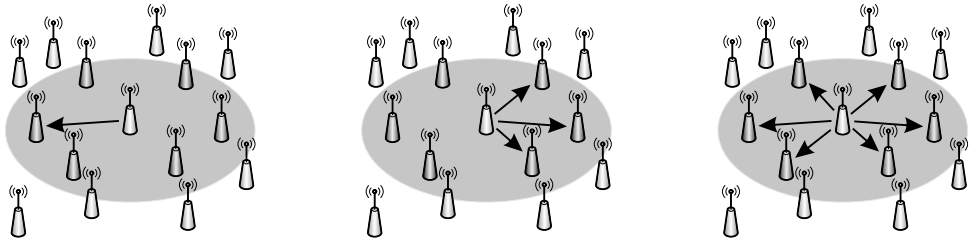
Moreover, the so-called *exposed terminal problem* is similar to the hidden terminal problem, and is defined herein for the sake of completeness (cf. Observation 2.9).

Definition 2.26 Exposed Terminal Problem. Suppose, a network consists of four nodes a, b, c, and d, as exemplified in Figure 2.4(b). Node b transmits a packet which is addressed to node a but not to node c. At about the same time, node c wants to transmit a packet directed only to node d. However, node c prevents its transmission using Carrier Sense (cf. Definition 2.8) right before its transmission: Both (directed) transmissions from node b as well as from node c are theoretically possible, since node a and node d would receive the corresponding packet without distortion. The Carrier Sense operation here wastes bandwidth and leads to needless waiting.

Observation 2.9. According to Definition 2.26, the exposed terminal problem just occurs due to the directed communication which is realized as *unicast* (or as *multicast*). Otherwise, the information sent by a node would be relevant for all of its one-hop neighbors. However, since the communication of our self-organizing protocol is only based on *broadcasts*, this problem is just of little importance within this work, and therefore will not be taken into consideration furthermore (cf. Section 5.2).

Definition 2.27 Unicast, Multicast, Broadcast. The *unicast* is a directed transmission from a sender towards one specific receiver (cf. Figure 2.5(a)). The *multicast* is a directed transmission from a sender towards a specific group of receivers (cf. Figure 2.5(b)). The *broadcast* is a transmission from a sender towards all potential receivers within its communication range (cf. Figure 2.5(c)). Apart from the broadcast, an addressing scheme is required in general.

Since the hidden terminal problem is inherent to multi-hop topologies, each node additionally requires knowledge about its hidden neighbors for an efficient and reliable communication – especially if the communication protocol is based on self-organization. The *constraint graph* $\vec{G}_C(i)$ is one formalization to describe the information needed by node $i \in N$.



(a) Unicast (b) Multicast (c) Broadcast

Figure 2.5: Examples for the communication between nodes. A gray shape represents the communication range of its centric sender, arrows represent directed links.

Definition 2.28 Constraint Graph. The *constraint graph* $\vec{G}_C(i) = \big(N_C(i), E_C(i)\big)$ of a node $i \in N$ is a directed subgraph of the corresponding network graph $\vec{G} = (N, E)$ with the set

$$N_C(i) = \{i\} \cup N_1(i) \cup N_2(i) \tag{2.4}$$

of nodes (i.e., vertices) and the set

$$E_C(i) = \big\{(j,i) \in E \colon j \in N_1(i)\big\} \cup \big\{(k,j) \in E \colon \big(\exists j \in N_1(i) \colon k \in N_1(j)\big)\big\} \tag{2.5}$$

of corresponding links (i.e., edges).

Observation 2.10. For each node $i \in N$ in a single-hop topology, i.e., in a complete graph, holds: $N_C(i) = N$.

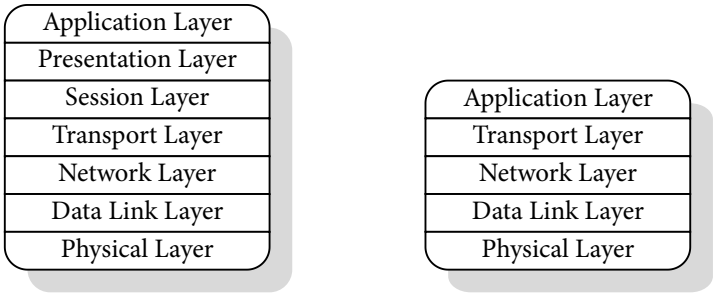
Since multi-hop topologies are hard to monitor, we introduce a special type of sensor node called *sniffer*.

Definition 2.29 Sniffer. A *sniffer* is a *passive* sensor node (cf. Section 2.4.5). That means that this node does not transmit data but instead listens to the shared medium and records all data received within a certain period of time. Hence, in any topology, a sniffer does not have any outgoing edge. Therefore, a sniffer does not interfere with the communication of nearby nodes, but only observes its (one-hop) neighborhood.

Observation 2.11. Apart from collisions, the (one-hop as well as two-hop) neighborhood of a sniffer is identical to that one of an "active" node which transmits messages to the network.

According to Definition 2.2, all participants of a Wireless Network communicate through a shared medium. Nodes compete for this shared medium by means of a multiple access strategy. These competitions can cause packet collisions which are undesired since they violate real-time constraints, destroy information, and waste information and energy for potential retransmissions. Furthermore, a collision could even result in the loss of possibly unique information. Therefore, an efficient and collision reducing operation of such multiple access networks is necessary. Theses demands are met by the main service[3] of our self-organizing protocol for WSNs: the *Medium Access Control* (MAC).

[3] See Section 8.2 for further feasible services.



(a) The 7-layer model of the OSI network architecture (cf. [201])

(b) The reduced WSAN protocol stack as suggested in [5]

Figure 2.6: Two types of protocol stacks commonly used for wireless communication.

Definition 2.30 Medium Access Control (MAC). A *Medium Access Control* (MAC) protocol manages when and how each participating transceiver may access the shared communication medium in order to avoid (or at least to reduce the chance for) packet collisions. Consequently, this would reduce information loss and temporal as well as energy efforts for retransmissions. To optimize the multiple access to the shared medium, a MAC protocol may provide services like *addressing schemes* and *channel access control mechanisms*.

According to the 7-layer model of the *Open Systems Interconnection* (OSI) network architecture (cf. Figure 2.6(a)), Medium Access Control is part of the data link layer [201, 87]. In WSNs as well as WSANs, there is usually no strict implementation of all seven OSI layers to increase the runtime efficiency as well as to reduce the implementation effort. Instead, a reduced protocol stack is suggested by Akyildiz et al. in [6, 5] (cf. Figure 2.6(b)):

- On the one hand, the boundaries of the OSI layers may overlap. For instance, today's RF chips keep getting more powerful, e.g., some of them provide hardware address checking (cf. Chipcon's CC1100 radio transceiver [175]). Therefore, the assembled hardware already implements some services of several OSI layers, e.g., the physical layer, and actually parts of the data link layer (hardware address checking).

- On the other hand, just a reduced set of layers may be implemented. Since common sensor nodes nowadays still have just low performance and limited memory (cf. Definition 2.6), a full implementation of all OSI layers would be a significant overkill for most application scenarios and hardware platforms (cf. also [157][4]).

To prevent packet collisions due to the concurrent medium access, there are several protocols available for managing and controlling the multiple access to the shared communication medium. In general, MAC protocols can be characterized on basis of several aspects – to name but a few: For instance, the *control* of the protocol may be *centralized* at a single controller or it may be *distributed* amongst all network components. Next, the transmission

[4]Diploma thesis conducted in conjunction with this work.

schedule may be *static* without the chance for change or it even may be *dynamic* for a flexible adaption, e.g., on topology dynamics. Furthermore, the transmission of messages within the network may range from *best-effort* to a *guaranteed* delivery. Moreover, the intention of the protocol may be based on contention-based *Carrier-Sense Multiple Access* (CSMA) or on schedule-based *Time-Division Multiple Access* (TDMA) – which is the most commonly used classification.

Definition 2.31 Carrier-Sense Multiple Access (CSMA). A contention-based *Carrier-Sense Multiple Access* (CSMA) protocol is a probabilistic MAC protocol supporting the best-effort strategy. Right before its transmission, the sender tries to verify that within its communication range no other node is currently transmitting a radio packet. For this purpose, each sender first listens to the communication channel. Next, it uses the information from its radio receiver unit to decide, whether the desired channel is busy or not. If there is no observable signal, the node finally starts its own transmission. Otherwise, the channel is considered to be busy. In this case, most CSMA protocols use a (probabilistic) *back-off algorithm* to postpone the retry, and to consequently reduce the probability of concurrent retransmissions on the shared communication medium. Amongst others, representatives of CSMA protocols for WSNs are for instance B-MAC [143], S-MAC [197], and T-MAC [185, 186].

Definition 2.32 Time-Division Multiple Access (TDMA). A schedule-based *Time-Division Multiple Access* (TDMA) protocol divides the communication channel into non-overlapping *time slots*. Each node has assigned one (or more) individual time slots for transmission, i.e., within the node's communication range no other node is allowed to transmit a radio packet at the same time on the same channel. The implementation of equally sized slots ensures fairness and the transmission at a fixed frequency bounds message latency. Amongst others, representatives of TDMA protocols for WSNs are for instance ISOMAC [199], LMAC [187], PEDAMACS [42], and TRAMA [148, 149].

Observation 2.12. Of course, hybrid MAC protocols implementing features from CSMA protocols as well as from TDMA protocols also are feasible. Amongst others, representatives of such hybrid MAC protocols for WSNs are for instance Crankshaft [77], HashSlot [22, 14], WiseMAC [58], and Z-MAC [154].

Such a schedule-based assignment supports a guaranteed data rate since it minimizes the probability of collisions. However, this requires a consistent network *coordination* among the nodes.

Definition 2.33 Coordination. In the field of network protocols, *coordination* describes the (temporary) process of harmonizing the actions of a set of network components to reach a common goal, e.g., collision free medium access. Besides, coordination is a special form of interaction (cf. Section 2.1).

The coordination of a common network schedule can be achieved in (a combination of) different ways:

Global Clock A global clock offers a uniform and accurate time base for the network components. This global clock has to be provided by a dedicated node or an external device, but would also allow to globally define the start of each time slot (cf. TRAMA [149]).

However, such a centralized approach for time synchronization always involves a single point of failure. Furthermore, to synchronize on the global clock, additional hardware may be required at the nodes (cf. Section 8.2.1). Thus, relying on a global clock does not meet our demands (cf. Section 1.2)

A priori knowledge Each node obtains (e.g., during the installation phase) a priori knowledge about the scheduling of its time slots (cf. Z-MAC [154]). Based on this knowledge, there may be further negotiation and competition of the nodes about time slots. However, the nodes may be incapable to handle topology dynamics satisfactorily – unless adaptive scheduling techniques are made available as required. As a consequence, a fixed schedule which is based on a priori knowledge is much too rigid for our purposes (cf. Section 1.2).

Self-organization The coordination about the time slots among the nodes is managed by the nodes themselves in a self-organizing manner without explicit time synchronization (cf. Definition 2.1). This approach does not rely on a central authority, but the ongoing interaction of the nodes makes it very robust against topology dynamics. However, additional costs for communication and computation have to be considered.

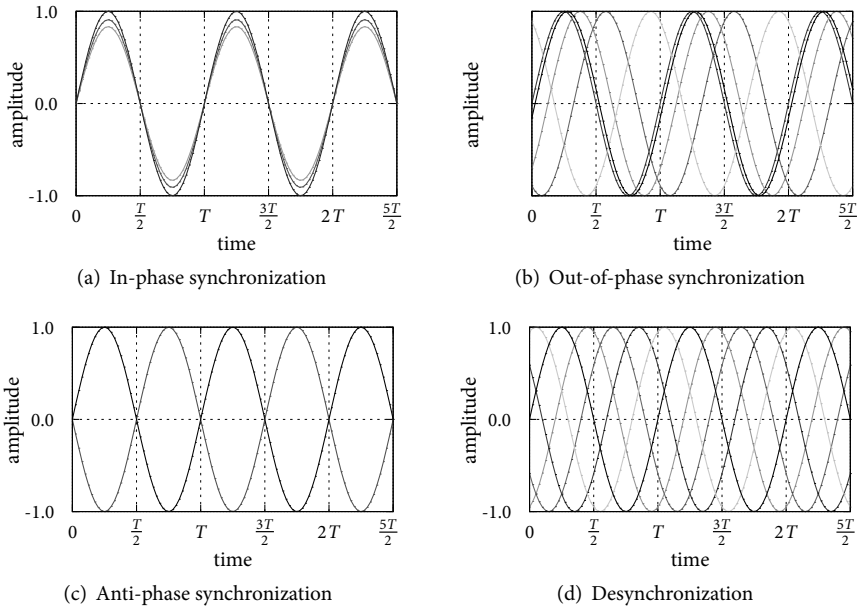
Hybrids In fact, combinations of the aforementioned coordination approaches are also feasible.

We intend to be able to react flexibly on topology dynamics (cf. Section 1.2). Therefore, we do not want to rely on a global clock or on a priori knowledge. Instead, we do want to use a decentralized approach. In particular, we prefer a self-organizing approach within this work.

Nevertheless, and as indicated before, time plays a major role for TDMA protocols as well as for coordination. As a consequence, *time synchronization* is an important coordination service for some WSN applications, e.g., to obtain data consistency (*sensor fusion*) or to maintain a certain TDMA protocol (*coordinating communication*) (cf. Section 8.2.1).

Definition 2.34 Time Synchronization. *Time synchronization* describes the process to provide a common notion of time across a distributed system like a Wireless Sensor Network.

Some WSN applications rely on this explicit understanding of a common time base. For instance, time synchronization is required to measure the velocity of mobile object, to detect snipers, to analyze seismic activities, or to monitor volcanic eruptions (cf. [192]). Moreover, it also may be used for coordination (cf. Definition 2.33), for communication (cf. for instance TDMA protocols), or for routing purposes (cf. Section 8.2.2). Furthermore, such a common notion of time may also support energy saving (e.g., in the form of coordinated sleeping phases) and sensor reachback (cf. [108]). Therefore, a more or less accurate time synchronization service has to be installed. For this reason, several algorithms and protocols regarding the service's establishment, maintenance, and accuracy yet exist. Examples include the following protocols: *Reference Broadcast Synchronization* (RBS) [59], *Timing-Sync Protocol for Sensor Netowrks* (TPSN) [69], *Time-Stamp Synchronization* (TSS) [114, 156], *Flooding Time-Synchronization Protocol* (FTSP) [111], and *Tight Time Synchronization* (Tiny-Sync) [198]. Further approaches and protocols for time synchronization for Wireless Networks are surveyed for instance in [48, 153, 158, 164], and for more complex networks relying on the pulse-coupled oscillator framework (cf. Section 3.2) in [9, 134, 161, 194] but also in Section 3.3.



(a) In-phase synchronization

(b) Out-of-phase synchronization

(c) Anti-phase synchronization

(d) Desynchronization

Figure 2.7: Examples for certain forms of synchronous dynamics (cf. Definitions 2.35 to 2.38).

Later on, we will use periodic oscillators with identical frequency as clock model. Hence, we distinguish between different forms of synchronization (cf. Figure 2.7): *in-phase synchronization*, *out-of-phase synchronization*, *anti-phase synchronization*, and *desynchronization*.

Definition 2.35 In-Phase Synchronization. Within this work, *in-phase synchronization* describes the phenomenon of the phase difference between two (or more) oscillators with identical frequency being zero. That means, the oscillators are in unison since their periods start over at the same time. Figure 2.7(a) exemplifies the in-phase synchronization of three sinusoids (with varying amplitudes for enhanced visualization).

Definition 2.36 Out-of-Phase Synchronization. Within this work, *out-of-phase synchronization* describes the phenomenon of the phase difference between two (or more) oscillators with identical frequency not being equal to zero (or to an integer multiple of a common period). That means, the oscillators are not in unison since their periods always start over at different times. Figure 2.7(b) exemplifies the out-of-phase synchronization of five sinusoids.

Definition 2.37 Anti-Phase Synchronization. *Anti-phase synchronization* is a special type of out-of-phase synchronization. Within this work, anti-phase synchronization describes the phenomenon of the phase difference between two oscillators with identical frequency being half their period. As a consequence, assuming the phase difference of two sinusoidal oscillators being half their common period, the sum of both oscillator amplitudes always equals zero (so-called *destructive interference*). Figure 2.7(c) exemplifies the anti-phase synchronization of two sinusoids.

Definition 2.38 Desynchronization. Within this work, *desynchronization* is another special type of out-of-phase synchronization. Therefore, the phase difference between two (or more) oscillators is not equal to zero. In particular, desynchronization describes the phenomenon of the phases of all oscillators within the oscillator's constraint graph (cf. Definition 2.28) having a maximum temporal distance towards each other within the common period. This means for a complete network of $|N|$ desynchronized oscillators (e.g., the nodes of a single-hop topology) that the phase difference between two succeeding oscillators always equals the $|N|$-th part of the common period (cf. Definition 2.28, Observation 2.10, and Lemma 4.5). Thereby, the oscillators interleave and occur in a round-robin schedule.

In contrast, there is no general statement about the phase difference of two succeeding oscillators of a non-complete network (e.g., the sensor nodes of a multi-hop topology). One reason here is the potentially differing size of the individual constraint graphs (cf. Lemma 4.11). Figure 2.7(d) exemplifies the desynchronization of five sinusoids.

Observation 2.13. If the complete network consists of just two oscillators, desynchronization and anti-phase synchronization result in the same behavior.

Since desynchronization provides the basic idea of our self-organizing TDMA protocol for WSNs, we will have a closer look at this phenomenon in Part II. To begin with, we give a short introduction of potential analysis techniques in the next section.

2.2 Analysis Techniques

In general, there exist several techniques to analyze certain phenomena and to study a specific system. The most obvious one is the system itself to be subject to studies and examinations. However, this is often too expensive or not applicable at all. Instead, a model of the system has to be analyzed. Depending on this representation, the following techniques are feasible:

- the verification of empirically derived predictions by means of suitable *experiments* on a physical model of the system,
- the *analytical solution* of a mathematical model of the system, for instance in terms of a system of linear equations, and
- the execution of a sufficient number of *simulation* runs based on a certain mathematical model of the system, i.e., a valid simulation model.

Furthermore, all these techniques feature advantages as well as disadvantages. Depending on the objectives, the data set, and the available resources, each technique is able to efficiently cover just a distinct field of applications: Assuming, it is too costly or even impossible to create a physical model of the system of investigation, the only way out are mathematical models then – resulting in analytical solutions or corresponding simulations.

The Proof of Convergence in Section 4.3.1 but also [49, 112] give a first impression about the complexity of a corresponding mathematical (non)linear system. As a result, the design of an analytical solution of the primitive of desynchronization seems to be an inappropriate task and thus not very promising. Indeed, some researchers prefer simulations to get valuable results quite easy and fast. In contrast, [169] emphasizes the need for real-world testbeds, i.e.,

simulations are not always meaningful enough. Due to the conditions and circumferences of the system to be analyzed, we want to benefit from both techniques. Consequently, we focus on *experiments* and *simulations* within this work. In this regard, we follow [125], which supports the legitimate coexistence of simulations and real-world testbeds.

Before we describe the tools applied in this work in Section 2.4 and in Section 2.3, we briefly introduce the analysis techniques which are relevant for this work: Namely, *real-world testbed* as physical model of a system in Section 2.2.1, and *simulation* as popular usage of a mathematical model of the system in Section 2.2.2.

2.2.1 Real-World Testbed

A *real-world testbed* makes use of components which behave sufficiently similar (*replica*) or even identical (*original*) to that one of the real system. Ideally, a testbed operates under real-world conditions to draw valid conclusions. For this purpose, components of the testbed are subject to arbitrary but realistic phenomena, like people moving around or interfering systems nearby. To keep the experimental setup manageable but to still get meaningful results, a testbed usually represents the system on a small scale, i.e., it consists of just a small number of system components. Despite a potentially small scale of a real-world testbed, its outcome still may allow for interpolation and extrapolation. Also characteristic for real-world testbeds are specific equipment and devices for monitoring and logging purposes (cf. Section 2.4.5).

Benefits

On the one hand, the deployment of a real-world testbed has several benefits: Such an experiment is embedded in an environment of the real world, hence the hardware as well as the software are subject to realistic conditions. Next, all measurements, which are taken during experimental runs, are subject to conditions of the real world. Therefore, we do expect the results to be realistic as well. Moreover, using a testbed, it is possible to set up specific situations and scenarios including side effects, parallel processing, and concurrent events (e.g., switching off certain nodes). Finally, even though the real-world testbed could be small and simple, it helps to draw sound conclusions (using interpolation and extrapolation) about the system – mainly due to the realistic environment.

Limitations

On the other hand, the applicability of a real-world testbed is limited: Since a testbed uses real hardware, its realization may be quite costly. In addition, the extent of the testbed as well as the used hardware for execution and administration of the tests directly influences these costs. Next, there is no speed up during the test procedure in general. For instance, assuming a certain period length $T = 1\,\mathrm{s}$ in real will usually also consume 1 s in the experiments. Moreover, due to the system's concurrent processing and further unexpected side effects, especially distributed testbeds (as needed within this work) are hard to monitor and hard to control on-line. Sometimes, conclusions may be drawn just after the experiment (*ex post*), which may delay the development process and also could impact the testing efforts. Finally, due to the dynamics of the environment, it is virtually impossible to reproduce the outcome of a real-world testbed – even for identical setup conditions.

2.2.2 Simulation

A *simulation* tries to draw quantitative conclusions about a system. Thus, a simulation can be considered as the experimentation with different input parameters of a mathematical model. The objective is to calculate (e.g., performance-related) effects on the system behavior and the system's outcome (cf. textbooks on simulation, like [104, 160, 78, 98]).

For this purpose, a simulation always drives a proper simulation time. This simulation time denotes the real-time of the simulated system. Thus, the progress of this simulation time needs not necessarily be equal to the simulated real-time or to the elapsed runtime of the whole simulation. Each simulation model has several entities or components, whose attributes in total determine the state of the model at a certain simulation time. A set of transformation rules defines the state transition as well as the changes of the entities' attributes.

Depending on the underlying simulation model, there are two major types, namely *continuous simulation* and *discrete simulation*: Simulations based on continuous models calculate the state transition by differential equations and alike to get continuous results. Simulations based on discrete models compute the state transition in discrete time, i.e., within a countable set of points in time. These discrete points in time are the ones at which an *event* occurs. The state of a discrete system only changes at such an event. Consequently, a discrete-event simulation allows to skip entire time segments – till the next simulation event will occur.

Benefits

The installation of a simulation has several benefits: The node hardware may be too expensive or too error-prone to deploy an experimental setup. This node hardware is not required for simulation runs in general. Next, a certain scenario realized as (single) simulation run usually is much faster than the same scenario performed by real hardware, i.e., there is no one-to-one relationship between simulation time and simulated time. Furthermore, each simulation run is repeatable, i.e., simulations running with identical values for the (setup) parameters as well as for the (pseudo) *Random Number Generator* (RNG) will result in an identical outcome. Finally, and contrary to any real-world scenario, a simulation tool may provide the opportunity of *global knowledge*[5]. This global knowledge offers new and comfortable possibilities for controlling, monitoring, and deeper analysis.

Limitations

However, the creation or execution of an adequate mathematical model may require considerable computational power. For instance, most supercomputers are installed to support extensive physical simulations like weather forecast, explosion of nuclear bombs, and artificial neurons of a human cerebral cortex. In addition, the execution of a simulation is not always appropriate: First of all, every simulation model is always just an abstraction of the real world. Therefore, each simulation model has to be validated accordingly. Nevertheless, there remains the risk of biased results. Especially, when the simulator applies idealistic assumptions or an ideal environment, conclusions drawn from such simulations may be not sound or insufficiently detailed.

[5] The quality of the global knowledge is limited by the capabilities (e.g., available memory) of the simulation environment.

2.2.3 Summary

In this section, we characterized the theoretical aspects of promising techniques to analyze a certain phenomenon and to study a specific system. On the one hand, a real-world testbed (cf. Section 2.2.1) demonstrates its strength on emulating a WSN under real but adverse conditions, like node failure, noise, and unreliable links. On the other hand, a software simulator (cf. Section 2.2.2) may not only save time, but also may generate fully observable as well as reproducible scenarios. Moreover, the integration of a (pseudo) Random Number Generator (RNG) makes simulations controllable – notwithstanding the probabilistic component. Furthermore, simulations allow the creation of particular scenarios – which are hard to deploy as real-world testbeds. For instance, it is difficult to provoke the hidden terminal problem (cf. Definition 2.25) with hidden nodes starting up simultaneously at the very same time (i.e., within the same microsecond).

As both techniques do have benefits as well as limitations, simulations and real-world testbeds should be used in complement to stimulate each other. This postulation also is stated in [125]. Therefore, we will use these two techniques for our further analysis. In this regard, we will describe our self-developed simulation framework, which performs our simulations in Section 2.3. In Section 2.4, we will present our sensor node hardware, which is used within our real-world testbeds.

2.3 Simulation Framework

Since Wireless Networks and Wireless Sensor Networks in particular are complex systems, we want to access the advantages of simulation (cf. Section 2.2.2) for our studies on the settling process of a Wireless Network implementing our communication protocol. Since a huge number of network simulators with specific objectives and different scopes already exist, we first motivate the development of our own simulator in Section 2.3.1. Next, we describe the functionality and features of our self-developed simulator in Section 2.3.2. In Section 2.3.3, we specify the underlying simulation model. To create simulation models more comfortably, we also developed a script, which is introduced in Section 2.3.4.

2.3.1 Motivation

The benefits of simulations obviously are the fast and cheap creation of experimental setups (cf. Section 2.2.2). Especially, when the system to be analyzed consists of lots of entities, like (Wireless) Networks in general and Wireless Sensor Networks in particular. Therefore, various network simulation tools with different scope and objectives are already available as (commercial) off-the-shelf products. Well-known network simulation tools are *ns-2* [113], *OMNeT++* [139], and *EstiNet* [62] – to name but a few.

ns-2

The discrete event driven network simulation tool ns-2 [113, 31, 64] allows the (traffic) analysis of several network protocols and routing mechanisms. It implements just fundamental protocols of several layers of the protocol stack (cf. Figure 2.6). Most famous ones are *Hypertext Transfer Protocol* (HTTP) and *Transmission Control Protocol* (TCP).

The core of the ns-2 simulator is the event scheduler: It monitors the simulation time and triggers the events of the event queue, i.e., the corresponding network component consuming this event is set active. Each active component uses the event scheduler to issue events for packet handling. Moreover, all network components wait to consume such an event to further process a packet. The simulations for the ns-2 simulator are programmed in C++ and OTcl, which is an object oriented extension of the *Tool Command Language* (Tcl). It is used to initialize the event scheduler and to set up the network topology, e.g., when a component is sending a packet.

OMNeT++

Like ns-2, OMNeT++ [139, 189] is also a discrete event driven network simulation tool for (communication) networks. However, OMNeT++ is not a network simulator itself, but provides an *Integrated Development Environment* (IDE) for simulations. This network simulation platform combines several modules and frameworks for network analysis. Hence, OMNeT++ is also generally applicable to queuing networks, hardware architectures, and business processes. For instance, the so-called *INET framework* contains models for communication protocols of the Internet, like TCP, *Internet Protocol* (IP), and *User Datagram Protocol* (UDP).

The simulation kernel and class library of OMNeT++ is written in C++. This kernel uses the basic simulation classes to manage the simulation. Simple modules and components, respectively, are programmed as *Network Description* (NED). Using NED, such modules can be combined to create compound modules and thus larger components. Consequently, the whole network itself is a compound module then.

EstiNet

The commercial network simulation tool EstiNet [62, 191, 61] indeed is a network simulator and estimator. It supports all layers of the reduced WSAN protocol stack (cf. Figure 2.6(b)). Moreover, there are also add-ons for specific network types, like Mobile Ad hoc Networks and Vehicular Ad hoc Networks with just a limited representation of mobility.

In comparison to the aforementioned simulation tools, EstiNet provides a user-friendly *Graphical User Interface* (GUI) to design the network to be analyzed. For modeling purpose, several network infrastructure devices are available, like host, hub, switch, and router. In addition to modeling, this GUI is also to be used for monitoring, simulating and even debugging. Moreover, EstiNet can be turned from simulator to emulator, i.e., a simulated network (component) can interact with a real-world network (component). This means that the network within the emulator is assembled by physical and simulated components (cf. *Hardware in the Loop* (HIL) simulation).

Resumé

Sophisticated and powerful network simulation tools, like ns-2, OMNeT++, and EstiNet, already exist, but do focus on "classical" network issues, like routing behavior, data rate, network throughput, and channel utilization. In contrast, the main focus of our analysis

in Chapter 7 within this work is on the settling process of a Wireless Network implementing our self-organizing MAC protocol EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, respectively (cf. Chapters 5 and 6).

In particular, we are interested in whether, how, and when a certain scenario will desynchronize (cf. Definition 4.3). As a result, there is no need for a deep packet analysis. Thus, the "classical" network issues are far less important for this work. Consequently, the named network simulation tools do not suit our requirements. Therefore, we developed in Java a lightweight simulation tool which exactly meets our specific demands. This simulator EXTDESIMC supports our EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocols and is described in the next section.

2.3.2 Simulator extDeSIMc

In Section 2.3.1, we motivated our decision not to use one of the available and powerful network simulation tools, but to develop a new simulation tool ourselves instead. Our simulator EXTDESIMC just supports *ex post facto* visualization, and thus a retrospective analysis. Nevertheless, the simulated time of our simulator has to support a large timescale of high precision. Like our nodes framework (cf. Section 2.4), EXTDESIMC has to support a (simulated) precision of at least 1 µs. This high resolution implicates the handling of large timescales up to billions[6] of microseconds.

One efficient way to cover such a large timescale is the event-driven simulation approach (cf. [120]). This approach is implemented not only by the powerful network simulation tools mentioned in Section 2.3.1, but also by our simulator EXTDESIMC. In particular, our simulator maintains an event queue which contains all events sorted by their occurrences in ascending order. For a consistent timeline, the times of the events have to increase monotonically, i.e., new events never may be a thing of the past.

State Transition

Each event will be processed by a specific event handler. In general, this event handler corresponds to a certain node, which consumes this event and will generate a subsequent event. Thus, each event represents a certain state transition of a node. To enforce a certain simulation scenario and simulation setup (cf. Section 7.2), some events (in particular `DEAD`, `ON`, `OFF`, and `FIRING`) may be set in advance within the simulation model. The corresponding state diagram, i.e., the possible states plus the associated transitions, is depicted in Figure 2.8.

For each simulation holds that every node starts either in `ON` state or in `FIRING` state – in accordance to what is specified within the simulation model: The `ON` state represents the power-on of a node at the event time. Usually, this will be the first state for a node. Next, after its initialization, this node will immediately switch to the `PREPARE_FIRING` state. However, to enforce certain simulation scenarios, e.g., concurrently firing nodes, a node may start in the `FIRING` state. This means that the corresponding event starts up this node, and after initialization, the node will skip the `PREPARE_FIRING` state and immediately switch to the `START_FIRING` state. As a result, this node will broadcast an (empty) firing packet at that

[6] For instance, 50 min equals 3 billions of microseconds, i.e., 3 000 000 000 µs.

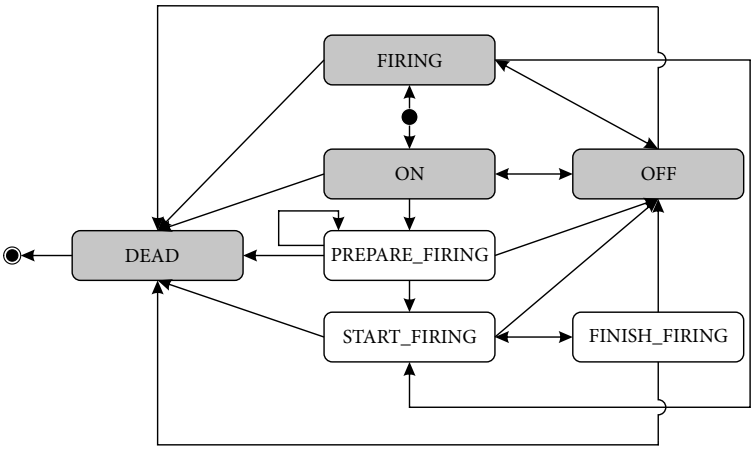


Figure 2.8: The state diagram of a simulated node in our extDeSIMc simulator. Times of the events to enter the states colored in gray have to be set in advance within the simulation model.

proper event time – regardless of a common understanding (cf. Section 5.8) and protocol specifications.

In the PREPARE_FIRING state, a node prepares its next firing and its next firing packet, respectively, e.g., it may collect neighbor information as described in Section 5.8.1. Depending on that particular listening strategy, the node either will remain in the PREPARE_FIRING state or it will switch to the START_FIRING state after a certain time, i.e., after few periods, as recommended in Section 5.8.1.

As further investigated in Section 5.4.1, the transmission of a (firing) packet takes some time. To reflect this fact, we introduced two distinct states (and events, respectively): The START_FIRING state indicates the start of the transmission, whereas the FINISH_FIRING state represents the end of the transmission. The time lag between the states START_FIRING and FINISH_FIRING depends on the declaration of the communication delay as well as the size (i.e., the number of neighbor information with regard to Section 5.6) of the corresponding firing packet within the simulation model. Consequently, the firing packet of a simulated node is delivered to appropriate and available[7] receivers when entering the FINISH_FIRING state. In addition, within this state the node will also plan its next time of firing (cf. Section 4.1). In particular, it plans the occurrence of its next event to enter the START_FIRING state. The underlying specific implementation of this generic framework for desynchronization (cf. Sections 4.3 to 4.6) as well as the realization of further practical issues (cf. Section 5.8) define the next time of occurrence of the START_FIRING state for a node.

To simulate an arbitrary and temporary power down of a node, e.g., due to low battery, we introduce the OFF state. The corresponding event moves the node into some sort of "sleep mode". Such a node may be powered on again using the corresponding event to enter the either the ON or the FIRING state . No more reactivation is allowed in the DEAD state, as this state

[7] A node in state OFF or DEAD is not available as receiver.

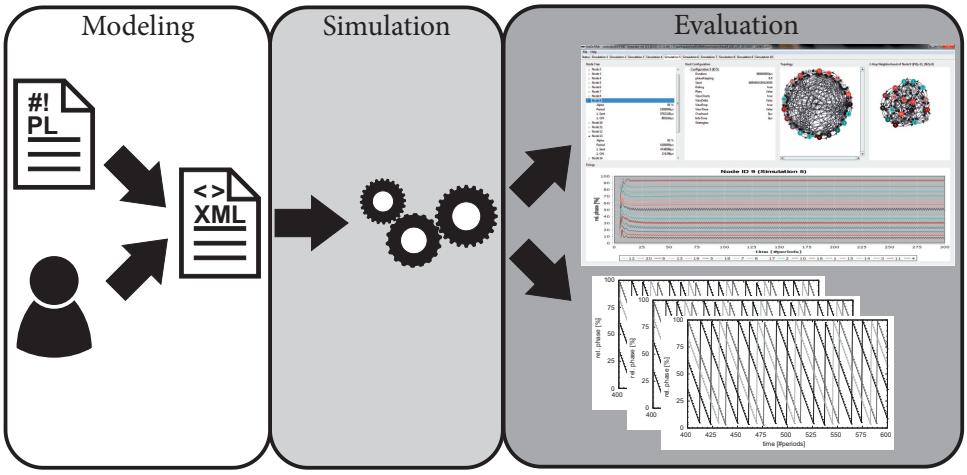


Figure 2.9: The process of a simulation run using our extDeSIMc simulator.

definitely removes a node from the network, and thus from the further simulation process. All these events, which enable a node to enter one of the states `ON`, `OFF`, `FIRING`, and `DEAD`, have to be specified in advance within the simulation model. Expect from the `DEAD` state, a node may switch to the `OFF` state from any other state. And a node may switch to the `DEAD` state from any other state at any time.

The simulation model specifies in advance not only the events regarding the power on as well as power down of the node. In addition, the end of the simulation also has to be specified in advance within the simulation model. This means that each simulation run ends after that predefined duration.

Simulation Process

The process of a simulation run using our self-developed simulation tool extDeSIMc is depicted in Figure 2.9. The whole process is mainly divided into the three steps *Modeling*, *Simulation*, and *Evaluation*:

First, the simulation model has to be specified as well-formed and valid *eXtensible Markup Language* (XML) file. This can be done manually by the user or automated per Perl script (cf. Section 2.3.4). We will describe the simulation model in detail in Section 2.3.3 and the Perl script in Section 2.3.4.

Next, such a simulation model then can be processed from within our extDeSIMc simulator. Since the simulation model may specify more than one scenario for the same topology, the user has to select the specific scenarios from the simulation model to be simulated. Afterwards, the simulator starts the simulation of these particular scenarios in parallel. During the execution, the simulator prints information[8] about the status of all simulation scenarios within a particular status tab. As soon as all user-selected scenarios have been finished, the

[8] The level of information can be specified within the simulation model.

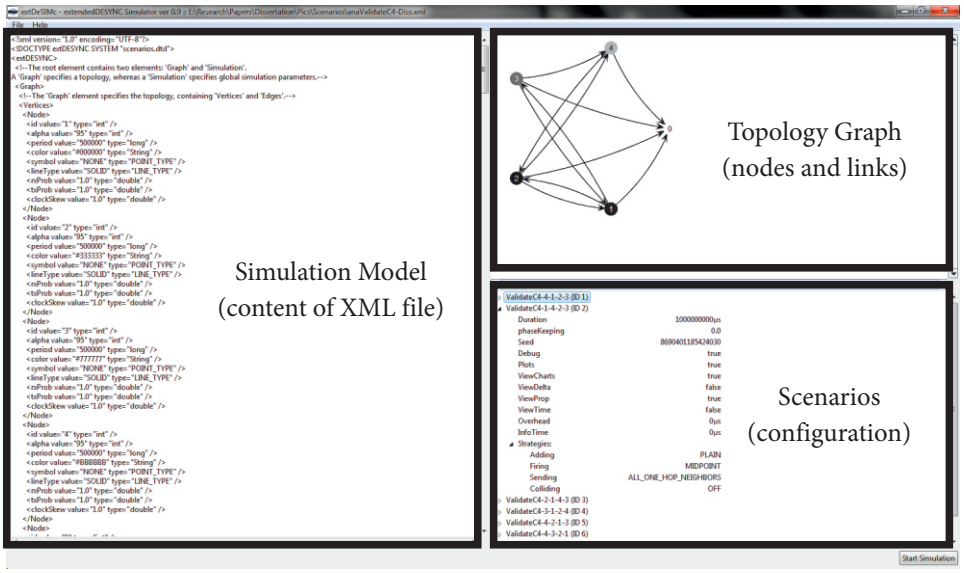


Figure 2.10: The start screen of our extDeSIMc simulator is split into three areas representing simulation model (left), topology graph (top right), and scenarios (bottom right).

outcome of these scenarios in retrospect finally may be visualized within the simulator's GUI (cf. upper right in Figure 2.9) and/or[9] as separate plots (e.g., GNU Plot) (cf. lower right in Figure 2.9). This supports a subsequent offline analysis, e.g., about the system behavior.

Moreover, the nodes in our real-world testbeds are able to log certain data at the serial interface. The output format of the node's logging complies with the input format of our simulator's visualization component. This is an important issue, since it offers not only an efficient visualization of real-world data but also a fast and smart way to cross-check the results of real-world testbed and the corresponding simulation for similar system behavior (cf. Section 7.1). Therefore, we are able to visualize the results of a real-world testbed and a simulation in an identical way.

Graphical User Interface

The start screen of our simulator (after reading in a simulation model file) is depicted in Figure 2.10: To control the simulation model in detail, the content of the currently imported XML file (including XML comments) is presented on the left. The topology graph as representation of the specified topology is depicted top right. Initially, all nodes are placed randomly on a circle layout. However, the nodes can be moved freely and for both, nodes and links, additional information is presented by tooltips. This topology is identical for all potential simulation scenarios listed on the bottom right area. Here, the user may check the

[9] The output format can be specified within the simulation model.

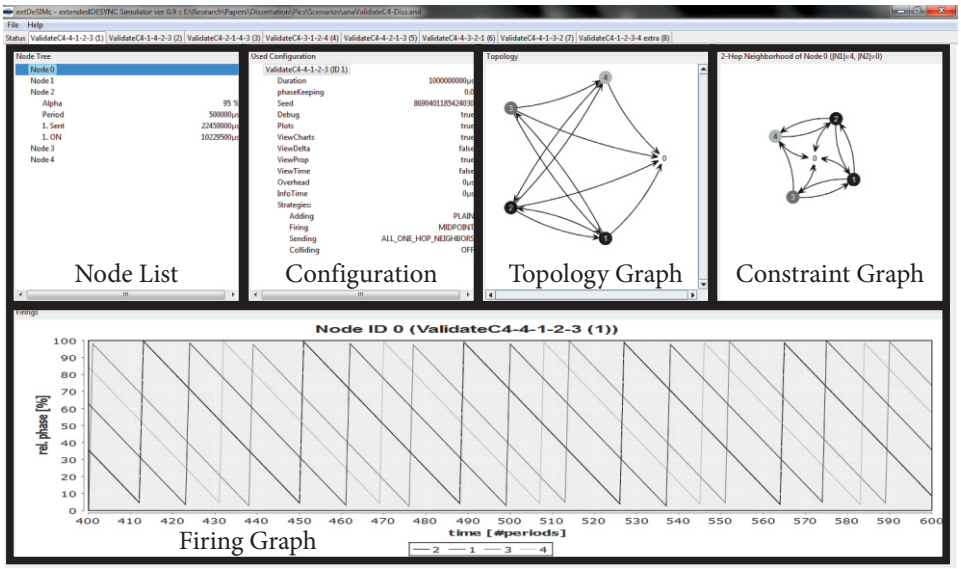


Figure 2.11: The simulation screen of our extDeSIMc simulator is split into five areas representing in the top half (from left to right) nodes list, configuration, topology graph, node's neighborhood, and in the lower half the node's firing graph.

parameters of the available scenarios. To start the actual simulation, at least one scenario has to be selected.

The simulation screen of our simulator provides a tabbed interface, i.e., the status tab and one scenario tab per simulated scenario. Each scenario tab is split into five areas as depicted in Figure 2.11: The top half is split again into four areas. Starting from left to right, the first area "Node List" in the top half lists all available nodes of the current scenario. Here, the user may select one particular node of interest. Consequently, the node selection affects the node dependent areas, namely "Constraint Graph" and "Firing Graph". In the next area "Configuration" on the right hand side, the parameters of the current configuration are listed. This information is also included in the simulation scenarios on the start screen (cf. Figure 2.10). The area "Topology Graph" to the right shows the graph of the current topology. This graph is identical for all other scenario tabs. Moreover, it also equals the one shown on the start screen (cf. Figure 2.10). Finally, the right most area "Constraint Graph" of the top half of the scenario tab presents the constraint graph (cf. Definition 2.28) of the node, which is currently selected in the "Node List". This visualization of the node's one-hop as well as two-hop neighbors supports the understanding of the node's firing behavior – especially for complex topologies. Similar to the topology graph of the start screen in Figure 2.10, the nodes of the "Topology Graph" as well as the "Constraint Graph" area can be moved freely.

The lower half "Firing Graph" of the scenario tab contains the firing graph from the point of view of the node, which is currently selected in the "Node List". In particular, the reception time of the node's neighborhood, i.e., its one-hop as well as two-hop neighbors, is plotted as

line chart. The domain axis of the firing graph denotes the time (in periods) since the simulation start. The range axis of the firing graph denotes the relative phase (here in percentage of the period length) of the received neighbor nodes' firings. Consequently, 100 % of a period is identical with 0 % of the subsequent period. The firing graph itself can be saved, printed, and zoomed (in and out). Since this display format facilitates the illustration of a node's state of desynchronization, such firing graphs will appear more often within this work.

2.3.3 Simulation Model

We tried to make the simulation model as abstract as possible and as realistic as necessary. Therefore, our simulator handles simulation models specified as well-formed and valid XML documents. The excerpt of such an XML file in Listing 2.1 exemplifies the description of a simulation model.

In principle, each model is partitioned into two main parts, namely one topology part `<Graph>` and one configuration part `<Config>` (cf. Listing 2.1): The topology part specifies the (static) topology graph, i.e., nodes (`<Vertices>`) and links (`<Edges>`). Each node `<Node>` is defined by several protocol-related parameters, like identifier, as well as by parameters of the presentation layer, like color or symbol (cf. Lines 6 to 12). Furthermore, a unidirectional link `<Link>` is represented by an edge, which is specified by one source node and one destination node as well as some quality parameters, like LQI (cf. Lines 16 to 20). Consequently, bidirectional links (cf. Definition 2.9) are specified by two corresponding `<Link>` elements.

To simulate different scenarios based on the same topology in parallel, the configuration part `<Config>` may contain one or more such simulation scenarios `<Simulation>` (cf. Lines 25 to 41). Hence, each scenario is running inside its own thread, and each scenario specifies its own stop criterion, i.e., its maximum simulation time (`<Duration>`). Furthermore, we integrated a simple Pseudo Random Number Generator into our simulator, which enhances the reproducibility of a simulation run by using the same random seed (cf. Section 6.4.2), which may be also specified per scenario. In addition, common initial values of several protocol-related parameters, like listening periods (cf. Section 5.8.1), as well as initial values of output-related parameters, like appearance of domain axis and range axis, may be set for each scenario. Moreover, each scenario contains a list `<EventList>` of predefined events `<Event>` (cf. Section 2.3.2). These events may enforce a distinct sequence by powering on (and powering down, respectively) the nodes from the topology part `<Graph>` (cf. Lines 34 to 38).

2.3.4 Generator Script

The simulation model may be quite complex and quite large – depending on the underlying topology and the number of defined scenarios (cf. Section 2.3.3). Hence, the manual model generation by the user may become a lengthy and error-prone task. Therefore, we developed a Perl script to semi-automatically create sound simulation models. Nevertheless, such a generated simulation model may be adapted or changed manually afterwards by the user.

However, the script prompts the user to enter values for the necessary parameters: First, the firing strategy has to be set. Here, the midpoint approach (cf. Section 4.3) would be set by default. Further, the user has to specify the network size, i.e., the number of nodes (default value: 25), the length of period T in µs (default value: 1 000 000 µs) as well as the

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE extDESYNC SYSTEM "scenarios.dtd">
<extDESYNC>
  <Graph>
    <Vertices>
      <Node>
        <id value="6" type="int" />
        <alpha value="0" type="int" />
        <period value="1000000" type="long" />
        <color value="#FFFFFF" type="String" />
        <!-- further parameters -->
      </Node>
      <!-- further nodes -->
    </Vertices>
    <Edges>
      <Link>
        <src value="5" type="int" />
        <dest value="6" type="int" />
        <lqi value="1.0" type="double" />
      </Link>
      <!-- further links -->
    </Edges>
  </Graph>
  <Config>
    <Simulation>
      <duration value="300000000" type="long" />
      <seed value="8690401185424030" type="long" />
      <createPlot value="true" type="boolean" />
      <firing value="MIDPOINT" type="FIRING_STRATEGY" />
      <listenPeriods value="3" type="int" />
      <phaseKeeping value="0.25" type="double" />
      <!-- further parameters -->
      <EventList>
        <Event>
          <type value="ON" type="EVENT_TYPE" />
          <time value="0" type="long" />
          <owner value="6" type="int" />
        </Event>
        <!-- further events -->
      </EventList>
    </Simulation>
    <!-- further scenarios -->
  </Config>
</extDESYNC>
```

Listing 2.1: Extract of a sample simulation model.

number of listening periods (default value: 1). Next, the user may specify the path to an available configuration, i.e., at least one sound `<Simulation>` element (cf. Section 2.3.3) has to be imported. Alternatively, the user may set the number of randomly generated simulation scenarios. The script will then generate an appropriate number of `<Simulation>` elements with default configuration parameters and one `ON` event for each node with event time set randomly within the first user-defined number of periods (cf. Section 7.2.3).

Further, the topology type (supporting only symmetrical, bidirectional links) has to be set. The script supports the forms from Section 2.1, namely complete graph (cf. Definition 2.19), star graph (cf. Definition 2.20), circle graph (cf. Definition 2.22), and line graph (cf. Definition 2.21), but also the forms binary tree (as a more complex and acyclic graph), dumbbell graph (cf. Section 7.4), and randomized graph (cf. Section 7.5). To create the latter form, the user has to enter a number indicating the average number of one-hop neighbors for each node. The script then randomly connects the nodes according to this value. Finally, the user can decide to automatically add a single sniffer node (cf. Definition 2.29) – or not.

2.4 Sensor Node Framework

In Section 2.3, we described our simulation framework. The outcome of a simulation based on sound simulation model allows quite cheap and fast conclusions about the system under certain conditions. However, each simulation model is just an abstraction, and thus a simplification of the real world.

Therefore, support documents and detailed information about the system's real-word behavior are required to create, to calibrate, and finally to verify a simulation model (cf. [125]). Especially, arbitrary and spontaneous phenomena, e.g., interference while people are moving around, are likely to appear in real world (cf. Section 7.2). Therefore, we deployed several real-world testbeds consisting of different types of sensor nodes to register these phenomena. Most of our testbeds were deployed at the Chair of Computer Engineering V at the Julius-Maximilians-University of Würzburg. Indeed, to distinguish nodes from each other, a unique identifier is assigned to each sensor node (cf. Sections 2.4.1 and 2.4.2).

Additionally, we also want to get detailed data from each node's point of view. One approach is to configure each node not only to act as network component, but also as measuring device, which logs certain data for on-line observation or for post-processing (cf. Section 2.3.2). Indeed, logging also affects the node's system load, i.e., the logging process takes noticeable time. However, the nodes of a real-world testbed will be configured identically in this regard, i.e., each node may be subject to the same additional time and effort. Altogether, this approach enables an efficient and cost-effective (*ex post facto*) inspection of the network. We will introduce our sensor node framework, including sensor node hardware and software, in the following sections.

2.4.1 SNoW^5 Sensor Node

Most of our testbeds are based on the modular and versatile SNoW^5 sensor node [20, 23]. This sensor node was designed and developed at the Chair of Computer Engineering V at the Julius-Maximilians-University of Würzburg and is depicted in Figure 2.12. The central unit of the SNoW^5 sensor node is the MSP430F1611 microcontroller [177] from Texas Instruments – a variant of the MSP430x1xx microcontroller family [174]. This 16 bit microcontroller provides 48 kB flash memory (ROM) and 10 kB RAM. However, the MSP430 allows to place variables in a specific memory location called *information memory* (infomem). We will use this part of the MCU's non-volatile memory to store fundamental configuration data, like node ID and settings for the RF unit (cf. Sections 5.6.1 and 6.4.1).

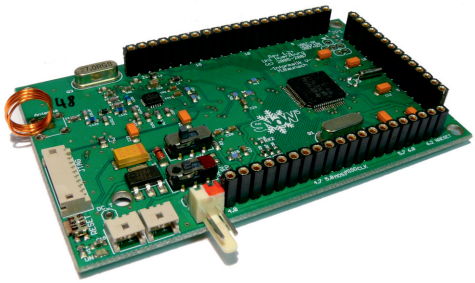


Figure 2.12: The SNoW^5 sensor node.

The MCU frequency of 8 MHz is offered by an external 8 MHz quartz crystal which provides an increased stability in comparison to a software adjustable Digitally Controlled Oscillator (DCO). As a result, a timer with resolution of 1 µs is made available. Among several bus interfaces like Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C) to connect sophisticated peripherals, this microcontroller also features analog-to-digital converter (ADC) as well as digital-to-analog converter (DAC) to read various types of sensors and to drive certain actuators, respectively. Both, the low computational power and the limited memory, make high demands on the implementation of our self-organizing protocol (cf. Chapters 5 and 6).

For wireless communication, the SNoW^5 sensor node is equipped with the sub–1 GHz RF transceiver CC1100 [175] from Texas Instruments. This powerful transceiver operates in half-duplex mode, i.e., the CC1100 transceiver can either transmit or receive a packet at the same time. The CC1100 is highly configurable, for instance it supports several software selectable modulation formats, several base frequencies depending on the current antenna circuitry as well as a basic hardware address check. As mentioned above, the settings of this RF transceiver are stored in the MCU's infomem. This transceiver as slave is connected via SPI bus with the central MCU as master. The convenient interrupt capabilities of this RF chip provide an appropriate base for highly precise TX/RX timestamping. This feature is quite beneficial when implementing a schedule-based MAC protocol. Therefore, it is noteworthy that two interrupt-indicating output pins of the radio unit are connected to interrupt-capable input pins of the central microcontroller at each node. With it, interrupts signaled by the radio unit can be handled by an appropriate *Interrupt Service Routine* (ISR) at the microcontroller. For instance, a TX FIFO underflow, the end of a SYNC word transmission, and the end of a SYNC word reception at the radio chip are able to trigger the corresponding interrupt at the microcontroller. This allows an enhanced coordination between both components and further enables a deeper timestamping (cf. [19] and Section 5.4.2).

For long-term data storage, the SNoW^5 sensor node has assembled Atmel's 16 Mbit non-volatile flash memory AT45DB161B [10]. There also exists a suitable embedded file system for this flash memory (cf. [109]). Initially, the SNoW^5 sensor node is not equipped with any

Figure 2.13: The eZ430 Chronos sensor node.

sensors and actuators – except for the RF unit[10]. However, almost all of the microcontroller's General Purpose Input/Output (GPIO) pins are available to plug appropriate peripherals, i.e., sensors or actuators. Furthermore, it features an RS-232 serial port communication interface for interaction with other devices, e.g., a *personal computer* (PC).

For wired programming purposes, the SNoW^5 sensor node implements the JTAG interface. This way of programming is quite costly and time-consuming as it requires direct (physical) access to the node. Hence, it is also possible (and more comfortable) to use a remote maintenance system like SNoW Ghost [13].

2.4.2 eZ430 Chronos

In contrast to the real-world testbeds consisting of the self-developed SNoW^5 sensor nodes, we also want to arrange real-world testbeds consisting of off-the-shelf sensor nodes. In this regard, we select the eZ430 Chronos [180] from Texas Instruments for the following reasons: First of all, the eZ430 Chronos provides the CC430F6137 [178]. The CC430 [179] is an MSP430 with an integrated RF core. However, an external 26 MHz quartz crystal drives this microcontroller. Hence, it is more tricky to provide a timer with resolution of 1 µs for the eZ430 Chronos than for SNoW^5.

As this CC430 microcontroller is an MSP430 microcontroller with an integrated sub–1 GHz RF transceiver, the eZ430 Chronos is compatible to the SNoW^5 sensor node to some extend. This compatibility allows the reuse of most of our software and applications already developed for SNoW^5 sensor nodes. Moreover, it also facilitates the establishment of a heterogeneous network consisting of SNoW^5 and eZ430 Chronos nodes (cf. [12][11]). As shown in Figure 2.13, the eZ430 Chronos is marketed as watch. As a result, it is highly portable and thus allows for real-world testbeds of high topology dynamics. In addition, this watch innately is equipped with several sensors, like pressure sensor and 3–axis accelerometer as well

[10] Even though a radio unit measures and manipulates a physical quantity, we consider the RF unit neither as sensor nor as an actuator within this work.
[11] Diploma thesis conducted in conjunction with this work.

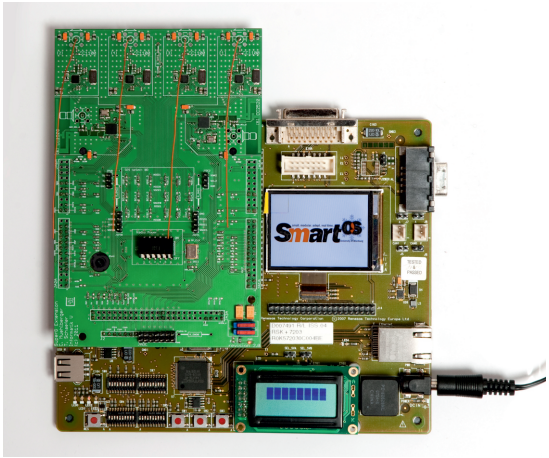


Figure 2.14: The SuperG gateway node, i.e., stacked on the RSK+ development board (olive-green) is the extension board (grass-green, on the left side) containing the RF units.

as a 96 segment *Liquid Crystal Display* (LCD) ready to be used. This supports the realization of several WSN applications.

Indeed, the eZ430 Chronos is not extensible, i.e., the connection of additional sensors or actuators is not intended. Furthermore, this commercial node offers a very limited number of communication interfaces. This further complicates debugging and monitoring of applications for the Chronos. Hence, the LCD may be used for a compressed data output to provide the desired information. This type of node and its unusual "debugging interface" is utilized in [12][12]: The objective of this work is the regulation of the protocol overhead of our MAC protocol by the self-organizing adaptation of the transmission power and thus the communication range.

2.4.3 SuperG Gateway Node

So far, we introduced two types of sensor nodes, namely SNoW^5 and eZ430 Chronos. These nodes are compatible with regard to the used MCU and RF unit, i.e., these nodes may communicate directly with each other. This already allows the analysis of additional interesting network scenarios and applications. However, to broaden the field of applications further by interconnecting yet existing or future generations of sensor nodes with differing hardware and communication protocols, we intended to integrate a more powerful node as gateway.

The SuperG gateway node was first introduced in [157][13]. In particular, this multilayer multiradio gateway node consists of an self-developed extension board for the *Renesas Starter Kit+* (RSK+) development board [150] for the SH7203 microcontroller [151] from Renesas and the RSK+ board itself as depicted in Figure 2.14. The extension board is equipped with four sub–1 GHz RF transceivers CC1100 [175] from Texas Instruments as well as two 2.4 GHz

[12] Bachelor thesis conducted in conjunction with this work.
[13] Diploma thesis conducted in conjunction with this work.

RF transceivers CC2520 [176] from Texas Instruments. The RF transceivers are connected as slave via SPI bus with the central SH7203 microcontroller at the RSK+ board as master. The placement of the CC2520 transceivers should make the SuperG gateway node ready for communication with ZigBee compliant sensor nodes, like MicaZ and TMote Sky (cf. Section 2.1).

As described in [129], the core element of this gateway architecture is its *Extended Distribution System* (EDS) – an extension of the *Distribution System* (DS) of the IEEE 802.11 WLAN infrastructure [86]: The EDS provides main services for distribution (i.e., message flow) and integration (i.e., protocol translation). To link different networks efficiently together, the EDS offers several operation modes, like router, hub, and switch, as well as a so-called *portal* per network as network-specific access point. In conjunction with the current operation mode, the SuperG gateway architecture not only allows protocol conversion (depending on the underlying MAC protocols), but also offers some sort of media conversion, e.g., if networks based on different frequency bands have to be interconnected.

2.4.4 SmartOS

In Sections 2.4.1 to 2.4.3, we described the sensor node hardware of our real-world testbeds briefly. Indeed, the utilization of an *operating system* (OS) is not mandatory to execute simple software on sensor nodes. Nevertheless, to run complex software of sophisticated applications or to coordinate time-critical processes like our self-organizing MAC protocol, an operating system becomes essential. Moreover, an operating system also supports and facilitates the development and reusability of software components, like drivers, modules, and packages. However, the operating system has to meet our demands on the development of a self-organizing MAC protocol for Wireless Sensor Networks, for instance a modular design and real-time operation. Therefore, we decided in favor for SmartOS – a small, modular, adapt, real-time operating system. Consequently, all our real-world testbeds (cf. Section 7.1) are based on SmartOS version 2.7.666. A minimal SmartOS application (i.e., kernel plus idle task) for the MSP430 MCU consumes 4 kB program ROM and 96 B RAM (cf. [18]).

The SmartOS operating system is introduced in [24] and explained in detail in [18]. It provides fully preemptive and prioritized tasks together with a collaborative resource sharing approach. To facilitate real-time operation, SmartOS offers a sophisticated time management with a local 64 bit timeline with a resolution of 1 µs. In addition, it implements an unified interrupt and resource concept with priority ceiling. This allows the creation of periodic tasks and a precise timestamping of internal and external events (cf. Section 5.4.2).

Moreover, a large inventory of drivers and modules for SmartOS already is available. This makes the application development much easier. For instance, there exists a math library for square root calculation (cf. [21]) as well as intersection point calculation (cf. [25, 127]), an implementation of a Pseudo Random Number Generator (cf. Section 6.4.1), and drivers for several sensors and actuators like acceleration sensor, stepper motor control, and ultrasound transducer (cf. [25, 18]). Furthermore, the lightweight MAC protocol SmartNET is also available (cf. [13, 18]): This wireless communication protocol provides an addressing scheme and supports timestamping. Therefore, we adapted the SmartNET protocol for our self-organizing MAC protocol extended-Desync and extended-Desync$^+$, respectively, in Section 5.6.1. Notably, our implementation approach can be made applicable to a much wider class of radio transmitters making just minor modifications (cf. Section 5.6.1).

2.4.5 Sniffer

Finally, we want to pick up again the concept of the *sniffer* (cf. Definition 2.29 as well as [125]). Since each sensor node has just a local view with just limited knowledge about the surrounding network and environment, the installation of a sniffer may offer deeper insight into the network's information flow. As mentioned in Definition 2.29, a sniffer just receives the packets from its one-hop neighbors, but does not transmit a packet itself. However, the sniffer may transfer data about the received packets to a personal computer for monitoring and controlling purpose. This enables a meaningful comparison of real-world results and simulation output. In fact, this valuable concept describes rather a role of a node than a type of node.

Nevertheless, the sniffer is still part of the network, and thus, it is exposed to the same (harsh) environmental conditions like the other nodes. This is the reason why the sniffer may also miss firing packets. Consequently, global and omniscient knowledge is not guaranteed. Depending on the sniffer's constraint graph, the exposure of a single sniffer may be sufficient to completely cover the whole (multi-hop) network.

Part II

Desynchronization

> Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.
>
> *Pierre de Fermat*

Abstract

This part characterizes *desynchronization* as biologically inspired primitive. Based on this primitive we describe in detail our self-organizing MAC protocol EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, respectively. The main focus of our TDMA protocol is on robustness against arbitrary topology dynamics. As we will show, a straightforward adoption without further improvement of the primitive of desynchronization as self-organizing MAC protocol for single-hop topologies is not sufficient for proper operation in multi-hop networks. Therefore, we present the development process from the single-hop MAC protocol DESYNC towards the extended MAC protocol EXTENDED-DESYNC and eventually to the EXTENDED-DESYNC$^+$ protocol. This extended multi-hop version is more versatile (i.e., not restricted to specific network types) and more robust against arbitrary topology dynamics than single-hop variations.

Chapter 3 first introduces the mathematical model of pulse-coupled oscillators. This mathematical model is used not only to formalize the primitive of desynchronization, but also to illustrate its applicability for WSNs. Chapter 4 presents existing realizations of the primitive of desynchronization as MAC protocol for single-hop as well as multi-hop topologies. Our approach of the multi-hop protocol EXTENDED-DESYNC is described in Chapter 5: This extension implements the *phase shift propagation* (PSP) – a mechanism which enables each node to gain information about its two-hop neighborhood autonomously. However, due to the associated *stale information problem* in multi-hop topologies, we had to make this first approach of a self-organizing MAC protocol more robust. Therefore, we introduced probability as an additional protocol extension to relax this problem: The eventually resulting EXTENDED-DESYNC$^+$ protocol is described in detail in Chapter 6.

Chapter 3

Desynchronization

Abstract

In this chapter, we take a closer look to the term *desynchronization* from Definition 2.38. Therefore, we first introduce the *pulse-coupled oscillator* (PCO) framework to synchronize oscillators in Section 3.2. Its application to synchronize wireless sensor nodes is described in Section 3.3. Section 3.4 motivates the focus shift from synchronization to desynchronization. Furthermore, we also identify the problems when applying desynchronization to Wireless Sensor Networks based on the PCO framework.

3.1 Introduction

As already stated in preparation of Definition 2.34, time synchronization is an important and also well-understood service[1] for several Wireless Sensor Network applications. Especially, series of measured data are meaningful only when tagged with corresponding timestamps. Moreover, the success of synchronized actions performed by distributed entities as concept of coordination (cf. Definition 2.33) also relies on time synchronization. However, "inverse" to the process of synchronization is *desynchronization* (cf. Definition 2.38). Indeed, the term *desynchronization* has different meanings – depending on the discipline. In the following, we give two examples to gain a first impression:

- In the research field of computational neuroscience, *desynchronization* describes the procedure when oscillators (in particular, the oscillatory activity of neurons) loose their initially in-phase synchronization (cf. Definition 2.35) due to miscellaneous influences. Such influences are for instance the modification of system values, occurring forces, or particular feedback from external components (cf. [145]).
- In [193], an attack scenario in Wireless Networks at transport layer (cf. Figure 2.6(a)) is also called *desynchronization*. In this scenario, an attacker permanently broadcasts messages with disordered sequence numbers. This will eventually force the receiver to do nothing but request the retransmission of seemingly missing packets all the time.

Therefore, we have to further improve Definition 2.38 within this chapter. The pulse-coupled oscillator (PCO) framework [118] by Mirollo and Strogatz is one popular mathematical model to cover emerging properties within a (fully-connected) network of oscillators. Similar to the periodical pulses of oscillators, the primitive of desynchronization relies on periodic transmissions. Hence, it seems appropriate to implement desynchronization as MAC protocol for WSNs on the basis of this framework.

[1] See Section 8.2.1 for related work on time synchronization in Wireless Networks.

3.2 Pulse-Coupled Oscillator Framework

Inspired by the emergence of synchrony in nature, Mirollo and Strogatz established in [118] a general framework of pulse-coupled oscillators. In particular, this framework was inspired by the biological system of male fireflies flashing at night-time in southeast Asia and by the model of self-synchronization of cardiac pacemakers as described by Peskin in [142]. This nature-inspired general framework of Mirollo and Strogatz relies on the following assumptions:

P1. The network consists of a set N of oscillators.

P2. Periodically, at the frequency $f = \frac{1}{T}$ with (potentially individual) period T, each oscillator *fires*, i.e., it emits an externally perceptible pulse[2].

P3. Each oscillator resets its phase immediately after the pulse emission. Certainly, the time required by each oscillator for a phase reset has to be short in comparison to the time between two consecutively emitted stimuli of each oscillator.

P4. Each oscillator is able to perceive pulses from other oscillators.

Definition 3.1 Globally Pulse-Coupled. The oscillators are *pulse-coupled* when the reception of a pulse fired by another oscillator may impact the receiver's behavior, e.g., the firing of its next pulse. If each oscillator receives the pulses fired by any other oscillator, i.e., the corresponding network graph is complete (cf. Definition 2.19), the oscillators are *globally pulse-coupled*.

The pulsatile coupling is an important characteristic of this framework: Each oscillator adapts the time of its next emission according to the received pulses (during its current period), i.e., according to its coupling with other oscillators. Hence, to synchronize globally pulse-coupled oscillators, Mirollo and Strogatz use just a simple adjustment function in [118]: Let the internal state x_i of oscillator $i \in N$ increase monotonically towards a threshold at $x_i = 1$. If the internal state x_i of oscillator $i \in N$ at time t is equal to 1, i.e., $x_i(t) = 1$, oscillator i is firing a pulse and subsequently resets its internal state x_i back to 0. Due to the pulse-coupling, for the subsequent internal state $x_j(t^+)$ of any other oscillator $j \in N \setminus \{i\}$ holds

$$x_j(t^+) = \min\left\{x_j(t) + \varepsilon, 1\right\}. \tag{3.1}$$

That means that either the next internal state x_j at time t^+ of the oscillator j is pulled up by a constant positive amount $\varepsilon > 0$, or the receiving oscillator j is even forced to immediately fire a pulse as well (i.e., $x_j(t^+) = 1$).

However, to prove the convergence to the stable state of synchronized oscillators, Mirollo and Strogatz additionally made the following idealistic assumptions (cf. [118]):

P5. There is no loss of pulses (or stimuli).

P6. Each pulse is detected instantly after emission, i.e., there is no delay in the message propagation.

[2] A pulse can be considered as *stimulus*, e.g., an electromagnetic signal, a burst, or a broadcast – as in our case.

P7. Noise is absent, i.e., there is not any other interfering signal to be misinterpreted as pulse.

P8. Each oscillator is sensitive for another pulse immediately after the detection of a previous pulse, i.e., there is no rest period.

P9. As tightening of assumption P2, all oscillators now perfectly feature the same frequency f, i.e., there is also no clock drift.

P10. All computations are "perfect", i.e., the result of a computation is available immediately with arbitrary precision (cf. *continuous mathematics*).

P11. The oscillators are globally pulse-coupled (cf. Definition 3.1).

Based on these (idealistic) assumptions P1 to P11, Mirollo and Strogatz prove in [118] that for any set N of oscillators and for almost[3] all initial states, a network of globally pulse-coupled oscillators eventually[4] becomes *synchronized* (cf. [118, 171]). Indeed, networks of PCOs under these idealistic assumptions are very unlike for real-world scenarios. Notwithstanding, several counterparts of such a synchronizing network[5] can be found in nature, for instance spiking neurons, flashing fireflies, chirping crickets, firing cardiac pacemaker cells, and the cycling of earthquakes (cf. [118, 171]).

3.3 Using PCOs to Synchronize WSNs

The pulse-coupled oscillator framework as described in Section 3.2 offers several benefits:

B1. The system behavior emerges solely from the interactions of the oscillators. According to Definition 2.1, this is one prerequisite for self-organization and thus enables a distributed control of the system.

B2. Due to the self-organizing nature, there is no need for a central coordinator determining the point in time when an oscillator has to emit a pulse. Such a central component could limit the performance of the system (*bottleneck*) or even stop the system's operation (*single point of failure*). Therefore, we support the absence of such a central control.

B3. Each oscillator operates on just locally available data, namely the received pulses. This allows each oscillator to react autonomously on each detected pulse.

B4. Therefrom, each oscillator can react fast on leaving or joining neighbors. This enables the system to scale well and to adapt fast to topology dynamics.

B5. Due to this adaptivity and fast reaction on topology dynamics, the entire system gets more robust against erroneous nodes and node failures.

[3] The set of initial states which will never synchronize has Lebesgue measure zero (cf. [118]).
[4] The set of initial states without any oscillator's adjustment also has Lebesgue measure zero (cf. [118]).
[5] At least the vast majority of the oscillators within such networks is synchronized.

Network type	Pulse-Coupled Oscillators	Wireless Sensor Networks
Connectivity	globally pulse-coupled	single-hop
Component	oscillator	(sensor) node
Stimulus	externally perceptible pulse	*firing* (radio broadcast)
Communication	full-duplex	half-duplex

Table 3.1: Comparison of Pulse-Coupled Oscillators with Wireless Sensor Networks.

B6. Each oscillator is based on a simple (cf. [118]) and just local rule set. Consequently, the computational task should be neither complex nor costly. This makes the pulse-coupled oscillator framework suitable for sensor nodes with just low computational power (cf. Definition 2.6).

Therefore, we exploit the benefits B1 to B6 of the self-maintaining pulse-coupled oscillator framework for (de)synchronization within the domain of Wireless Sensor Networks.

3.3.1 Adaptation

Some modifications of the framework from Mirollo and Strogatz in Section 3.2 are necessary for transferring pulse-coupled oscillators to Wireless Sensor Networks:

C1. We have to substitute sensor nodes for oscillators.

C2. Hence, each node has to emit periodically (with period T) some sort of stimulus which can be received by other nodes.

C3. Since all sensor nodes are able to interact wirelessly by definition (cf. Definition 2.6), and since a broadcast does not rely on a distinct target addressing scheme (cf. Definition 2.27), we will use broadcast messages as stimuli.

C4. The counterpart of a globally pulse-coupled network is the single-hop topology due to its completeness.

C5. Due to the hardware limitations of typical sensor nodes (cf. Definition 2.6), communication in a WSN is just in half-duplex mode.

Applying the modifications C1 to C5, the globally pulse-coupled (and idealistic) network of oscillators from Section 3.2 complies well to a fully-connected (and idealistic) single-hop topology of a Wireless Sensor Network. Table 3.1 briefly compares the pulse-coupled oscillator framework with Wireless Sensor Networks.

Definition 3.2 Firing, Firing Message. In order to distinguish clearly between a broadcast as periodical stimulus (as requested in Section 3.3) and an ordinary (but maybe sporadic) broadcast propagating another type of message, we call the periodical broadcast *firing* and the corresponding broadcast message *firing message*. This naming is in accordance with the PCO framework from Section 3.2.

3.3.2 Related Work

The PCO framework in Section 3.2 always aims to synchronize the pulses of all network components. Due to the importance of synchronization services for some applications, successful implementations of synchronization protocols for WSNs already exist. In the following, we present a selection of some related synchronization protocols.

Assuming the idealistic conditions P1 to P10 from Section 3.2 but excluding condition P11, Lucarelli and Wang implemented in [108] a decentralized synchronization protocol for multi-hop Wireless Sensor Networks. The synchronization update rules are based on the PCO framework from Section 3.2. Additionally, this framework utilizes nearest neighbor communication to handle time varying topologies. This enables the protocol to handle a non globally pulse-coupled network with certain topology dynamics like joining or leaving nodes. Therefore, this implementation offers synchronization under ideal assumptions for multi-hop networks, i.e., ideal networks without global coupling.

However, the optimal conditions from Section 3.2 are unlikely for real-world Wireless Sensor Networks: For instance, noise in real-world deployments is as undeniable as the absence of continuous mathematics at sensor nodes. Therefore, Hong and Scaglione implemented in [84] the PCO framework for a fully-connected WSN under more realistic conditions, i.e., this protocol considers packet loss, noise, and propagation delay. Furthermore, a *refractory period* (cf. Section 6.3) is integrated into this synchronization protocol to further stabilize the system: This rest period succeeds each reception of a firing message. It disables the reception of further firings for a specific amount of time and thus eliminates infinite feedback loops.

Finally, Werner-Allen et al. implemented in [192] a synchronization protocol based on the PCO framework from Section 3.2. Instead of immediate responses on each incoming impulse, each node first records all incoming pulses within one period and "reacts" all at once at the time of its next firing. As a consequence, this protocol is able to synchronize the nodes of a multi-hop network even under more realistic radio effects, like message delay or packet loss.

3.4 Using PCOs to Desynchronize WSNs

So far, we presented the PCO framework to synchronize a network of oscillators in Section 3.2 as well as a network of nodes in Section 3.3. However, in some natural systems, the main objective is not "synchronization" as exemplified so far. In fact, there are (biological) systems in which the system components do not synchronize but rather desynchronize according to Definition 2.38. Therefore, each system component tries to maximize the phase difference towards its neighbors resulting in a regular overall pattern. Indeed, such a desynchronizing system can be specified by (a variant of) the PCO framework.

For instance, different gaits of an animal can be emulated by a set of correspondingly pulse-coupled oscillators representing the animal's extremities (cf. [171]). Another, but also well-analyzed biological system is the male Japanese Tree Frog (*Hyla japonica*): It uses mating calls to attract female Japanese tree frogs. Aihara et al. experimentally observed in [2] that two male Japanese tree frogs anti-phase synchronize[6] their mating calls (with just little overlapping). Using the PCO framework, this behavior was mathematically formalized in [3]. In

[6] Please recall Observation 2.13 here.

addition, this mathematical model was further extended to desynchronize a complete network of three[7] alternately calling male Japanese tree frogs (cf. Section 4.5).

In all scenarios mentioned within this Section 3.4 so far, the components do not synchronize first and negotiate an appropriate phase pattern afterward. Instead, the system components use self-maintaining adjustment rules to generate the desired (de)synchronization pattern autonomously and right from the start. This makes these systems robust for topology dynamics, like leaving or joining beings or network components. This corresponds well with our focus on a self-organizing MAC protocol which is based on the primitive of desynchronization for multi-hop topologies without a central control and without explicit time synchronization (cf. Section 1.2). Therefore, we can enhance the adaptation of the PCO framework to a WSN from Section 3.3. Indeed, a firing message does not push the receiving nodes to fire in unison, but each node has to adjust its next time of firing in accordance to the received firing messages of its neighbors. Hence, based on this PCO framework, implementations of the primitive of desynchronization as MAC protocol for WSNs already exist. These implementations mainly differ

- in the algorithm, which estimates the next time of firing of a node,
- in the set of relevant firings, i.e., which received firings are considered for the estimation at each period, and
- in the intended field of application and type of topology.

In the next chapter, we give an overview on available implementations of such MAC protocols for Wireless Sensor Networks. However, we will see that all these protocols are either restricted to particular types of networks or are not robust against topology dynamics (cf. Section 1.2).

[7] The phase difference between two subsequent calling tree frogs equals $T/3$ then.

Chapter 4

Desynchronization as MAC Protocol

Abstract

After the introduction of the primitive of desynchronization, we present in this chapter various implementations of MAC protocols applying this primitive. As a start, we develop a generic framework which implements the primitive of desynchronization as MAC protocol for WSNs in Section 4.1. Based on this framework, we draw some conclusions regarding single-hop as well as multi-hop topologies in Section 4.2. Next, we describe some practical implementations: We begin in Section 4.3 with the *midpoint* approach, which is quite popular due to its simple computation. Furthermore, this approach will be the basis for our algorithm in Chapter 5 and Chapter 6, respectively. Next, we introduce the *local max degree* approach in Section 4.4. The approach depicted in Section 4.5 is inspired by the mating behavior of male Japanese tree frogs as mentioned before (cf. Section 3.4). Finally, the *artificial force field* approach, which is based on the pattern formation of mobile robots, is explained in Section 4.6.

4.1 Generic Framework

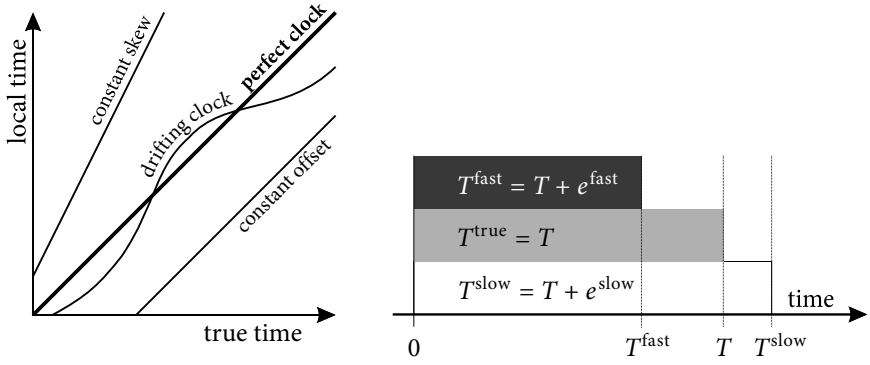
In this section, we introduce the generic framework implementing the primitive of desynchronization as MAC protocol for WSNs. For a more comprehensive formalization, we define a unique *identifier* (ID) $i \in \mathbb{N}^+$ for each (active) node (cf. Definition 2.29). Moreover, we do not further distinguish between the identifier and the node itself in the set N of (active) nodes, but, without loss of generality, let them be numbered consecutively $1 \leq i \leq |N|$. For a common understanding, we have to specify the *modulo* operation first, due to several conventions possible.

Definition 4.1 Modulo. Given the natural number $a \in \mathbb{Z}$ (called *dividend*) and the natural number $b \in \mathbb{Z} \setminus \{0\}$ (called *divisor*). The *modulo* operation $a \bmod b$ calculates the remainder r of the integer division a/b as

$$\begin{aligned} r &= a \bmod b \\ &= a - b \cdot \left(\operatorname{sign}(a) \cdot \operatorname{sign}(b) \cdot \left\lfloor \frac{|a|}{|b|} \right\rfloor \right) \end{aligned}$$

with

$$\operatorname{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}.$$



(a) Simple scheme of potential ratio of local clock time to true time.

(b) Examples for individual periods.

Figure 4.1: Exemplification of local clock and its impact on period T.

Consequently, the algebraic sign of the result r of the modulo operation within this work always equals the algebraic sign of the dividend a, i.e., $\text{sign}(r) = \text{sign}(a)$. Consequently,

$$-5 \bmod +3 = -2$$
$$\text{and also} \quad -5 \bmod -3 = -2 \quad \text{holds.}$$

This Definition 4.1 is in accordance with the implementation of Java 1.7.0 (cf. [75]) used for our simulator (cf. Section 2.3) as well as of the msp430-gcc 3.2.3 compiler used for the programming of the real-world sensor nodes (cf. Section 2.4).

Observation 4.1. For $|a| < |b|$ holds $a \bmod b = a$.

In compliance with the biologically inspired primitive of desynchronization (cf. Chapter 3), each node of the network has to fire (cf. Definition 3.2) periodically with a common period $T \in \mathbb{N}^+$ at the common frequency

$$f = \frac{1}{T}. \tag{4.1}$$

Since sensor nodes (and computers in general) are finite state machines, the measurement and thus the internal representation of the continuous physical quantity "time" within such digital systems is always discrete with $t \in \mathbb{N}^0$.

Apart, the common period T is not identical for all nodes due to individual clock drift and the resulting limitations in the frequency stability. Figure 4.1(a) shows a scheme of the potential ratio of a node's local clock time to the true time. Therefore, T^i denotes the period T measured with the local clock of node i. Consequently, $T^i = T + \gamma^i$ with a certain amount $\gamma^i \in \mathbb{Z}$ typically holds. This phenomenon is overdrawn in Figure 4.1(b) with $\gamma^{fast} < 0$ and $\gamma^{slow} > 0$.

Every time a node $i \in N$ completes its period, it broadcasts its firing packet, resets its phase immediately, and updates its *next time of firing* $t_i^+ \in \mathbb{N}^0$ based on its *current time of firing*

$t_i \in \mathbb{N}^0$. Similar to the notation used for a node's individual period, the term t_j^i denotes the firing time t_j of node j from the local point of view (i.e., registered with the local clock) of node i. Consequently, the meaning of the term t_i^i is always identical with the term t_i, i.e., $t_i^i = t_i$ always holds. If the context clearly specifies node i's point of view then we will mainly just use t_j instead of t_j^i for the sake of convenience. Nevertheless, this differentiation will be relevant for the exchange of timing information in Section 5.5.

Let $\phi(t_i, t) \in (-T, T)$ be the *phase shift* of a node $i \in N$ since its current firing at time t_i and a given point in time $t \in \mathbb{N}^0$ as

$$\phi(t_i, t) = \big(t - t_i\big) \bmod T, \tag{4.2}$$

utilizing the modulo operation from Definition 4.1. Please note that the phase shift $\phi(t_i, t)$ may be negative, which will be relevant in Section 5.3.

Observation 4.2. If $|t - t_i| \leq T$ always holds (cf. Observation 4.1), the phase shift has not to be normalized to the period T, i.e., $\phi(t_i, t) = t - t_i$ holds.

When node $i \in N$ receives a firing packet of its one-hop neighbor $j \in N_1(i)$, node i records the time of reception according to its local clock as time t_j. Using Eq. (4.2), node i is able to calculate the phase shift $\phi(t_i, t_j)$ towards this neighbor j. For example, $\phi(t_i, t_j) = 0.25 \cdot T$ means that node i has already finished a quarter of its current period when node j transmitted its firing packet at time t_j.

Knowledge about the (firing times of the) nodes of its constraint graph $\vec{G}_C(i)$ (cf. Definition 2.28) is sufficient for a node $i \in N$ to solve the hidden terminal problem in a self-organizing manner (cf. [49]). Therefore, node i relies on the (dynamic) set

$$N_R(i) \subseteq N_C(i) \setminus \{i\} \tag{4.3}$$

of *relevant nodes* to update its next time of firing t_i^+. That means, if a relevant node $j \in N_R(i)$ of node $i \in N$ changes its time of firing, then node i has to adjust its next time of firing t_i^+ correspondingly as a consequence. The elements of the set of relevant nodes depend on the practical implementation, i.e., the vague specification in Eq. (4.3) will be defined more accurately within each subsection describing the corresponding relation of the particular implementation. Anyway, a nonempty set $N_R(i) \neq \varnothing$ of relevant nodes always contains the *phase neighbors* of node i, i.e., its *predecessor* $p(i) \in N_R(i)$ as well as its *successor* $s(i) \in N_R(i)$. These phase neighbors will be of special interest.

Definition 4.2 Phase Neighbor, Successor, Predecessor. A node $j \in N_R(i)$ is called *phase neighbor* of node $i \in N$, if the firing of node j happens – from the (limited) knowledge of node i – just before or just after the firing of node i itself. Hence, we call the successive phase neighbor $s(i) \in N_R(i)$, which broadcasts its firing packet just after node i, *successor*, and the previous phase neighbor $p(i) \in N_R(i)$, which broadcasts its firing packet just before node i, *predecessor*. Formally stated,

$$s(i) = \operatorname*{argmin}_{s \in N_C(i) \setminus \{i\}} \phi(t_s, t_i) \qquad \text{and} \qquad p(i) = \operatorname*{argmin}_{p \in N_C(i) \setminus \{i\}} \phi(t_i, t_p). \tag{4.4}$$

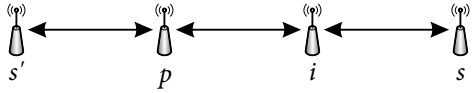


Figure 4.2: Within this line graph L_4, nodes s and s' are transmitting concurrently without any collision, i.e., $t_s = t_{s'}$ holds. In addition to node p, which is the predecessor $p(i)$ of node i, nodes s and s' both are candidates to be the successor $s(i)$ of node i.

Observation 4.3. From a nonempty set $N_R(i) \neq \varnothing$ of relevant nodes of node i directly follows the existence of the phase neighbors of a node $i \in N$. Thus, every node i with degree $d_i \geq 1$ has one predecessor $p(i) = j \in N_C(i) \setminus \{i\}$ as well as one successor $s(i) = k \in N_C(i) \setminus \{i\}$.

Observation 4.4. It may happen that successor and predecessor are the very same node, i.e., $s(i) = p(i)$.

Observation 4.5. In single-hop topologies the packets of concurrently transmitting nodes collide at all times. Therefore, the result of the argmin function from Definition 4.2 is injective and well-defined. In multi-hop topologies the packets of concurrently transmitting nodes may not collide. If there are nodes with identical transmission times in the set of relevant nodes, the result of the argmin function from Definition 4.2 may be not well-defined. Instead, the argmin function could return a set of identifiers (cf. Figure 4.2). Since our algorithm rather relies on the phase shift between nodes than on the node's ID, which is accessed just for a more comprehensive formalization, w.l.o.g. we will choose a node from the result set by random as successor and predecessor, respectively. Indeed, this random selection is valid as a smarter selection of successor (and predecessor, respectively) from such a result set has no real effect on the system behavior (cf. Lemma 4.8).

According to the primitive of desynchronization, each node tries to maximize the temporal distance of its time of firing towards its relevant nodes. Therefore, each node $i \in N$ uses an *adjustment function*

$$\varphi_i\left(N_R(i), t\right) \in (-T, T) \tag{4.5}$$

to determine its correction value at time $t \in \mathbb{N}^0$ depending on its current set $N_R(i)$ of relevant nodes. Using t_i in Eq. (4.5), this function computes the actual displacement of node i between its current time of firing t_i and its optimal time of firing. Finally, node i is able to set its next (absolute) time of firing t_i^+ depending on the result of its adjustment function, its current time of firing t_i, and the common period T as

$$t_i^+ = t_i + T + \varphi_i\left(N_R(i), t_i\right). \tag{4.6}$$

After some iterations, i.e., the so-called *settling phase*, each adjustment function converges to a (local) fixed-point. With it, the whole system is in the stable state of *(perfect) desynchrony*.

Definition 4.3 Desynchrony, Perfect Desynchrony. The system is in the stable state of *desynchrony*, if there exists a point in time t such that the adjustment function of each node $i \in N$ remains constant[1] for any later point in time $t^{++} > t$, i.e.,

$$\varphi_i\left(N_R(i), t^{++}\right) = \varphi_i\left(N_R(i), t\right). \tag{4.7}$$

The system is in the stable state of *perfect desynchrony*, if each node respects the same (temporal) distance to its relevant nodes, i.e., the adjustment function converges to 0: There exists a point in time t such that the system is in the stable state according to Eq. (4.7), and additionally the adjustment function of each node $i \in N$ for any later point in time $t^{++} > t$ results in

$$\varphi_i\left(N_R(i), t^{++}\right) = 0. \tag{4.8}$$

Once, the system is in the stable state of (perfect) desynchrony, the phase shift between a node $i \in N$ and any of its relevant nodes $j \in N_R(i)$ remains constant – apart from clock drifts and adaptations to topology changes (cf. Section 7.5).

4.2 General Conclusions

Solely based on the generic framework from Section 4.1 and independent of a specific implementation approach, we can already draw some conclusions. Therefore, and based on our initial enumeration in [122], we additionally identify the following observations regarding the primitive of desynchronization as MAC protocol. These observations are discussed for single-hop topologies in Section 4.2.1 and for multi-hop topologies in Section 4.2.2.

4.2.1 Single-Hop Topology

Since the network graph of a single-hop topology is complete (cf. Definition 2.23), the set N of nodes equals the union of any node $i \in N$ and its one-hop neighborhood $N_1(i)$, i.e., for each node $i \in N$ holds $N = N_1(i) \cup \{i\}$. Indeed, we assume the shared communication medium to be error-free, i.e., there is no packet loss due to noise or hardware defects. Hence, a packet transmission is successful, if there is concurrently not any other packet transmission. That means, just one single node of the complete network is allowed to transmit at any point in time. With it, we can draw the following conclusions for desynchronization in single-hop topologies:

Lemma 4.1 Single-Hop S1. *All nodes of a single-hop topology have the very same degree.*

Proof. According to Definition 2.23, the network graph of a single-hop topology is complete, and it consists of the set N of nodes. Therefore, for each node $i \in N$ holds

$$d_i = |N| - 1. \tag{4.9}$$

□

[1] Technically speaking, the nodes are out-of-phase synchronized (cf. Definition 2.36), if the adjustment function converges to a constant value being not equal to 0.

Lemma 4.2 Single-Hop S2. *If node $i \in N$ is phase neighbor of another node $j \in N_1(i)$, then node j in turn is the* opposing phase neighbor *of i. For instance, let w.l.o.g. node i be predecessor of node j, i.e., $p(j) = i$, then node j is successor of node i, i.e., $s(i) = j$.*

Proof. Node $i \in N$ is phase neighbor of node $j \in N_1(i)$. Let w.l.o.g. node i be successor of node j, i.e., $s(j) = i$. According to Definition 4.2, t_i is the smallest time for which $t_i > t_j$ holds. Assuming, node $k \neq j$ is the predecessor of node i, i.e., $p(i) = k$. According to Definition 4.2, t_k is the greatest time for which $t_k < t_i$ holds. Due to the completeness of the corresponding network graph of a single-hop topology (cf. Definition 2.23), node $k \in N_1(i) \cap N_1(j)$ holds. Hence, t_i is the smallest time for which $t_i > t_k$ holds. This is a contradiction to $s(j) = i$, where t_i is the smallest time which holds $t_i > t_j$. Therefore, $t_k = t_j$ must hold. If $j \neq k$, this would cause collisions at node i due to the single-hop topology. Moreover, this behavior is contradictory to our assumption that just one single node of the complete network is allowed to transmit at any point in time. Hence, $j = k$ must hold, and thus $p(i) = j$. □

Lemma 4.3 Single-Hop S3. *Moreover, node $i \in N$ is the opposing phase neighbor of its phase neighbors $j \in N_1(i)$ and $k \in N_1(i)$, i.e., node i is successor of solely one neighbor j as well as predecessor of solely one neighbor k.*

Proof. Assuming, node $i \in N$ has more than one successor (the case for more than one predecessor is analogous): In compliance with Definition 4.2, the firing time of each successor of node i is the smallest time which is greater than t_i. This means that for any two successors $s(i), s'(i)$ of node i with $s(i) \neq s'(i)$ has to hold $t_{s(i)} = t_{s'(i)}$. Using the argumentation from the proof of Lemma 4.2, $s(i) = s'(i)$ has to hold. This is a contradiction to our assumption that node i has more than one successor. Hence, node i has solely one successor $s(i) \in N_1(i)$. The rest of Lemma 4.3 follows directly from Lemma 4.2. □

Lemma 4.4 Single-Hop S4. *Every node $i \in N$ with degree $d_i \geq 1$ is always predecessor $p(j) = i$ and successor $s(k) = i$ of its phase neighbors $j, k \in N_1(i)$.*

Proof. Follows directly from Lemmas 4.2 and 4.3. □

Lemma 4.5 Single-Hop S5. *If the system is in the stable state of perfect desynchrony, the temporal distance between each pair of subsequently firing nodes then equals $T/|N|$.*

Proof. According to Definition 4.3, each node $i \in N$ of a system in perfect desynchrony respects the same (temporal) distance to its relevant nodes. Due to the primitive of desynchronization, the (temporal) distance of node i's time of firing towards each element of its set $N_R(i)$ of relevant nodes is maximized. Using Lemma 4.2, for the phase neighbors $s(i)$ and $p(i)$ of each node $i \in N$ holds $\phi(t_i, t_{s(i)}) = \phi(t_{p(i)}, t_i)$. Therefore, all $|N|$ nodes are distributed in temporal equidistance along the common period T, i.e., the temporal distance between each pair of subsequently firing nodes is equal to $T/|N|$. □

Lemma 4.6 Single-Hop S6. *For a collision-free communication within a single-hop topology the minimum required number of distinct time slots within period T equals the size $|N|$ of the network.*

Proof. The network graph of a single-hop topology is complete (cf. Definition 2.23). In conjunction with Lemma 4.1, for a collision-free communication the period T has to support at least

$$\max_{i \in N}\{|N_1(i)|+1\} \overset{(2.3)}{=} \max_{i \in N}\{d_i+1\} \tag{4.10a}$$

$$\overset{(4.9)}{=} |N|-1+1 \tag{4.10b}$$

$$= |N| \tag{4.10c}$$

distinct time slots. □

4.2.2 Multi-Hop Topology

Since the network graph of a multi-hop topology is not complete (cf. Definition 2.24), the hidden terminal problem (cf. Definition 2.25) is inherent to multi-hop topologies. Once more, we assume the shared communication medium to be error-free, i.e., there is no packet loss due to noise or hardware defects. Hence, a packet transmission is successful, if there is no other packet transmission at the same time within the communication range of the receivers. That means, two or more nodes of the system may be able to transmit a radio packet concurrently without any interference. With it, we can draw the following conclusions for desynchronization in multi-hop topologies:

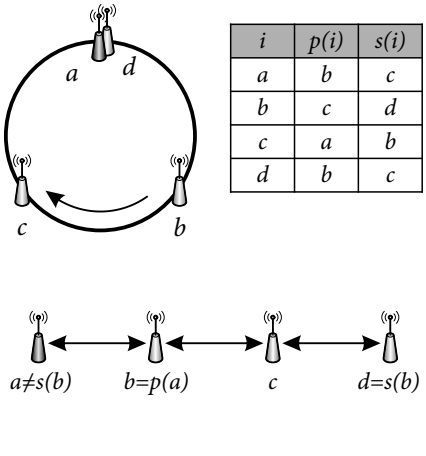
Lemma 4.7 Multi-Hop M1. *The degree of the nodes in a multi-hop topology may differ. However, for each node $i \in N$ holds $d_i \leq |N|-1$.*

Proof. According to Definition 2.24, the network graph of a multi-hop topology is not complete, but it consists of the set N of nodes. Therefore, for each node $i \in N$ holds $d_i \leq |N|-1$. The sample scenario in Figure 4.3(a) depicts different node degrees in a multi-hop topology: $d_a = 1$, but $d_b = 2$. □

Lemma 4.8 Multi-Hop M2. *If node $i \in N$ is phase neighbor of another node $j \in N$, then node j needs not to be the* opposing phase neighbor *of i. For instance, let w.l.o.g. node i be predecessor $p(j) = i$ of node j, then node j need not to be successor of node i in turn, i.e., $s(i) \neq j$ may hold. Instead, another node $k \neq j$ is successor $s(i) = k$ of node i then (cf. Figure 4.3(a)).*

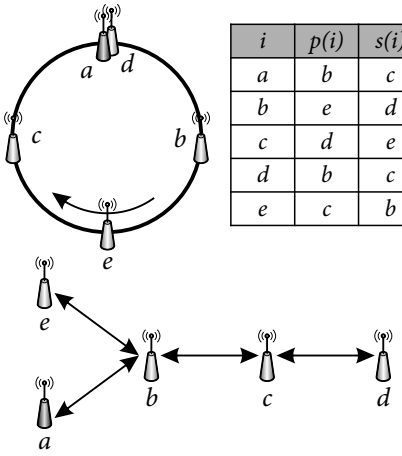
Proof. According to Definition 4.2, for both phase neighbors $s(j)$ and $p(j)$ of any node j holds $s(j), p(j) \in N_1(j) \cup N_2(j)$. On the basis of Lemma 4.7, let node $i \in N_2(j)$ be the predecessor $p(j) = i$ of node $j \in N_2(i)$, but for i's successor holds $s(i) \in N_1(i)$ with $s(i) \notin N_1(j) \cup N_2(j)$. Therefore, node j cannot be the successor of node i. Exemplified by the sample scenario in Figure 4.3(a), let $p(d) = b$ and $d = s(b)$ with $a \notin N_1(d) \cup N_2(d)$, i.e., node b is predecessor of node d, and node d is successor of node b. Moreover, for node a holds $p(a) = b$, i.e., node b is also predecessor of node a. However, node a is not successor of node b, which is node d. Therefore, node a is not the opposing phase neighbor of its own phase neighbor node b. □

For the next conclusion, we have to introduce the concept of a so-called *multiple phase neighbor*:



i	p(i)	s(i)
a	b	c
b	c	d
c	a	b
d	b	c

(a) Here, node $b = p(a)$ is predecessor of node a, node $d = s(b)$ is successor of node b.



i	p(i)	s(i)
a	b	c
b	e	d
c	d	e
d	b	c
e	c	b

(b) Here, node a is neither predecessor nor successor of any node, since node $d = s(b)$ is successor of node b, node $d = p(c)$ is successor of node c.

Figure 4.3: Example scenarios on the general conclusions on multi-hop topologies. In both examples, node a and node d send at the same time without causing any collisions.

Definition 4.4 Multiple Phase Neighbor. Assuming, node $i \in N$ is successor of all elements of a set $N_S(i) \subseteq N \setminus \{i\}$ of nodes as well as predecessor of all elements of (another) set $N_P(i) \subseteq N \setminus \{i\}$ of nodes at the same time. Consequently, all nodes within the union $N_S(i) \cup N_P(i)$ share the same phase neighbor i. If $|N_S(i) \cup N_P(i)| > 2$ holds, we call node i a *multiple phase neighbor*. Exemplified by the sample scenario in Figure 4.3(b), node b is a multiple phase neighbor since $N_S(b) = \{e\}$ and $N_P(b) = \{a, d\}$ with $|N_S(b) \cup N_P(b)| = 3 \geq 2$.

Observation 4.6. In general, a node i on its own is not able to discover whether it is a multiple phase neighbor, or not. However, based on just locally available information, as a start the node's degree d_i (cf. Definition 2.16) may be an indicator in this regard.

Lemma 4.9 Multi-Hop M3. *According to Observation 4.5, the* argmin *function from Definition 4.2 may return a set of nodes. As a result, the phase neighbor $s(i)$ and $p(i)$ may be not well-defined. Therefore, in multi-hop topologies a node $i \in N$ could be successor of all elements of a set $N_S(i) \subseteq N \setminus \{i\}$ of nodes as well as predecessor of all elements of (another) set $N_P(i) \subseteq N \setminus \{i\}$ of nodes at the same time. If node i is a multiple phase neighbor according to Definition 4.4, i.e., $|N_S(i) \cup N_P(i)| > 2$ holds, the whole system could be destabilized when this multiple phase neighbor i adjusts its next time of firing. This will eventually lead to fluctuations of transmission times.*

Proof. In consequence to Lemma 4.8, a node $i \in N$ needs not to be the opposing phase neighbor of its own phase neighbors $s(i)$ and $p(i)$. Indeed, yet another node $j \in N$ has to take over the role of at least one opposing phase neighbor (cf. Observation 4.3). Since each node always has a single successor as well as a single predecessor according to Definition 4.2

in combination with Observation 4.5, node j could be a multiple phase neighbor according to Definition 4.4. Now, let node j be such a multiple phase neighbor, i.e., $|N_S(i) \cup N_P(i)| > 2$ holds. If node j adjusts its next time of firing, all nodes within the union $N_S(j) \cup N_P(j)$ could be affected at once: Either, a node $k \in N_S(j) \cup N_P(j)$ also has to adjust its next time of firing according to the adjustment of its phase neighbor j and in compliance with the primitive of desynchronization. Or, node j is not a phase neighbor of node k anymore, i.e., $k \notin N_S(j) \cup N_P(j)$ holds, and node k has to adjust itself according to another node $l \in N_1(k) \cup N_2(k)$ with $l \neq j$. Exemplified by the sample scenario in Figure 4.3(a) again, node c is multiple phase neighbor of the other nodes a, b, and d, i.e., $N_S(c) \cup N_P(c) = \{a, b, d\}$. □

Lemma 4.10 Multi-Hop M4. *In multi-hop topologies there may be nodes, which are either only predecessors (but not successors) of other nodes, or which are only successors (but not predecessors) of other nodes, or which are none of both (neither successor nor predecessor). Moreover, a node $i \in N$ being not a relevant node of any other node may vary its next time of firing within a certain time interval not triggering the adjustment of any other node's next time of firing.*

Proof. The first part of Lemma 4.10 follows from Lemmas 4.8 and 4.9. For instance, the scenario shown in Figure 4.3(a) exemplifies that node d is just successor, but not predecessor, whereas node a is just predecessor, but not successor. Next, assuming node $i \in N$ is not a relevant node of any other node, i.e., for all nodes $j \in N$ holds $i \notin N_R(j)$. In consequence of Eq. (4.6), the time of firing of node i is irrelevant for any other node's time of firing, since node i is not relevant at all. Therefore, if node i varies its time of firing in a certain time interval, i.e., especially not to become a relevant node, its adjustment does not affect the adjustment of any other node's next time of firing. In the sample scenario shown in Figure 4.3(b), node a is neither successor nor predecessor of another node. □

Lemma 4.11 Multi-Hop M5. *If a multi-hop system is in the stable state of perfect desynchrony, the temporal distance between each pair of subsequently firing nodes may not equal $T/|N|$ anymore. Moreover, the temporal distance between a node $i \in N$ and its phase neighbors $s(i)$ and $p(i)$ may now differ from the temporal distance between another node $j \in N$ with $j \neq i$ and its phase neighbors $s(j)$ and $p(j)$.*

Proof. The first part of Lemma 4.11 directly follows from Lemma 4.7. Next, assuming nodes $i, j \in N$ with $d_i \neq d_j$. From $|N_1(i)| \neq |N_1(j)|$ and in accordance with Eq. (4.3) could follow $|N_1(i) \cup N_2(i)| \neq |N_1(j) \cup N_2(j)|$. In the worst case, each node within the constraint graph of node i has a unique time of firing, i.e., for any pair of nodes $k, l \in N_C(i)$ holds $t_k \neq t_l$. If the same holds for node j, then the temporal distance between node i and its phase neighbors may be different to the temporal distance between node j and its phase neighbors. □

Lemma 4.12 Multi-Hop M6. *Besides, in multi-hop topologies there could be two (or more) nodes within the node set $N_C(i)$ of the constraint graph of a node $i \in N$ sharing the very same time of firing without causing interferences at all (cf. Definition 2.26).*

Proof. Let two nodes $j, k \in N_C(i)$ of node $i \in N$ with $j \neq k$. Assuming, $j \notin N_1(k) \cup N_2(k)$ and $k \notin N_1(j) \cup N_2(j)$ holds, i.e., both nodes are more than two hops away from each other. Therefore, $t_j = t_k$ may hold without interference (cf. Definition 2.25). Exemplified by the sample scenario in Figure 4.3(a) again, this would hold for nodes $a, d \in N_C(c) = \{a, b, c, d\}$ with $t_a = t_d$. □

Protocol	Year	Approach	$N_R(i)$	Topology	References
DESYNC	2007	Midpoint	$p(i), s(i)$	single-hop	[50, 49]
Scattering	2007	Midpoint	$N_1(i)$	single-hop[2]	[73]
Frog Call	2009	Frog-Call Inspired	$N_1(i)$	single-hop	[132, 131]
M-DESYNC	2009	Local Max Degree	D_i	acyclic multi-hop	[92]
DESYNC-ORT	2011	Midpoint	$p(i), s(i)$	single-hop	[173]
V-DESYNC	2012	Midpoint	$p(i), s(i)$	single-hop	[159]
DWARF	2012	Artificial Forces	$N_1(i)$	single-hop	[39]
M-DWARF	2012	Artificial Forces	$N_1(i) \cup N_2(i)$	multi-hop	[38]
EXTENDED-DESYNC	2009	Midpoint	$p(i), s(i)$	multi-hop	[128, 121, 122]
EXTENDED-DESYNC$^+$	2012	Midpoint	$p(i), s(i)$	multi-hop	[124, 126]

Table 4.1: A short comparison of several self-organizing MAC protocols implementing the primitive of desynchronization.

Lemma 4.13 Multi-Hop M7. *For a collision-free communication within a multi-hop topology the sufficient number of distinct time slots within period T equals* $\max_{i \in N}\{|N_C(i)|\}$.

Proof. In general, a sufficient number of time slots for firings to support a collision-free communication within the network may be less or equal to the size $|N|$ of the network. However, although all nodes share the same communication medium, nodes which are more than two hops (cf. Definition 2.13) away from each other could also transmit their firings concurrently without causing a collision, thus reducing the number of required slots. From a local perspective of a node $j \in N$, each element of the node set $N_C(j)$ of the constraint graph of this node $j \in N$ requires a distinct time slot within period T. In the worst case, node $j \in N$ has the highest the cardinality $|N_C(j)|$ of its node set $N_C(j)$ within the network, i.e., $|N_C(j)| = \max_{i \in N}\{|N_C(i)|\}$ holds. Therefore, for a collision-free communication within a multi-hop topology the sufficient number of distinct time slots within period T is equal to $\max_{i \in N}\{|N_C(i)|\}$. Since the minimum number of distinct time slots within period T may be even smaller according to Lemma 4.12, the given threshold is just an upper bound. □

Observation 4.7. As for single-hop topologies, each node of a (multi-hop) star topology requires its own, distinct time slot within period T for a collision-free communication. Therefore, at least $|N|$ time slots are required.

Observation 4.8. In consequence of Observation 4.7, if the system is in the stable state of perfect desynchrony, the temporal distance between each pair of subsequently firing nodes of a star topology also equals $T/|N|$ (cf. Lemma 4.5).

Observation 4.9. For any star topology holds $i = s(p(i))$ and $i = p(s(i))$ (cf. Lemma 4.2).

In the following Sections 4.3 to 4.6, we describe a selection of specific implementations of this generic framework for desynchronization. Table 4.1 overviews these implementations for a node $i \in N$ with focus on the realization

- of its set $N_R(i)$ of relevant nodes,

[2] Using an additional central unit, even acyclic multi-hop topologies are supported.

- of its adjustment function $\varphi_i\left(N_R(i), t\right)$, and
- of the type of topology for which the particular MAC protocol was intended to operate.

4.3 The Midpoint Approach

Due to its small set of nodes to be considered and its simple mathematical calculations, the *midpoint* approach is straight forward and easy to use. Hence, these characteristics make this approach feasible for the operation in constrained embedded systems and thus quite popular in WSNs. For instance, the first implementation of the primitive of desynchronization as MAC protocol DESYNC [50] for Wireless Sensor Networks utilizes this approach. Moreover, our robust MAC protocol for multi-hop topologies EXTENDED-DESYNC$^+$ (cf. Chapter 6) is also based on this approach.

To update its next time of firing, node i just relies on the data of its phase neighbors, i.e., its predecessor $p(i)$ and its successor $s(i)$ (cf. Definition 4.2). Therefore, the set $N_R(i) = \{p(i), s(i)\}$ of relevant nodes of node i has at most these two members. According to the generic framework from Section 4.1 and to maximize its temporal distance towards its relevant nodes, each node i aims on the midpoint of its phase neighbors' time of firing. This adjustment towards the average would lead the system to the desynchronized state in which all nodes fire at the midpoints of their phase neighbors' time of firing. For this purpose, node i has to capture the reception times $t_{s(i)}$ and $t_{p(i)}$ of the corresponding firings from its phase neighbors. With it, node i can calculate the (relative) phase shifts $\phi(t_i, t_{s(i)})$ and $\phi(t_{p(i)}, t_i)$, respectively. Hence, node i is able to compute its *adjustment factor* ε_{t_i} with respect to its current time of firing t_i as

$$\varepsilon_{t_i} = \frac{\phi(t_i, t_{s(i)}) - \phi(t_{p(i)}, t_i)}{2} \tag{4.11a}$$

$$\overset{(4.2)}{=} \frac{\left(t_{s(i)} - t_i\right) \bmod T - \left(t_i - t_{p(i)}\right) \bmod T}{2}. \tag{4.11b}$$

Observation 4.10. If Observation 4.2 is applicable, Eq. (4.11b) simplifies to

$$\varepsilon_{t_i} = \frac{\left(t_{s(i)} - t_i\right) - \left(t_i - t_{p(i)}\right)}{2} \tag{4.12a}$$

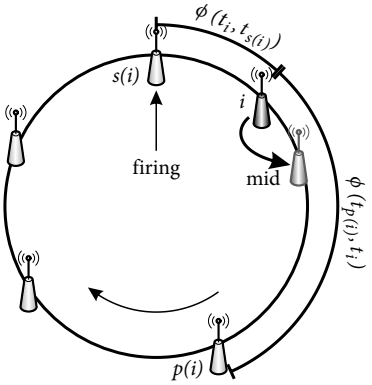
$$= \frac{t_{s(i)} + t_{p(i)}}{2} - t_i. \tag{4.12b}$$

The adjustment factor specifies the actual difference between the current time of firing of node i and its optimal time of firing at this stage. With it, the adjustment function from Eq. (4.5) at time t_i equals

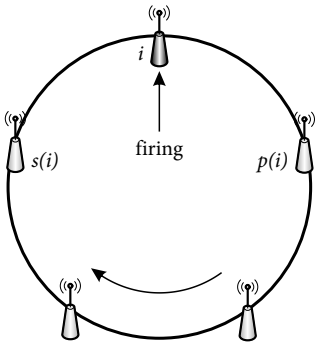
$$\varphi_i\left(N_R(i), t_i\right) = \alpha \cdot \varepsilon_{t_i} \tag{4.13a}$$

$$\overset{4.11}{=} \alpha \cdot \frac{\phi(t_i, t_{s(i)}) - \phi(t_{p(i)}, t_i)}{2}. \tag{4.13b}$$

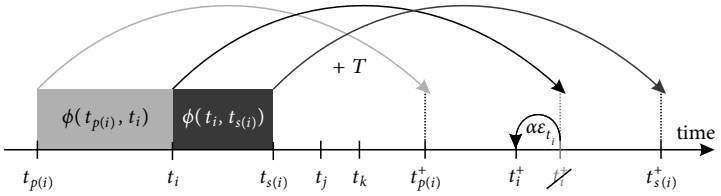
The *jump size parameter* $\alpha \in [0,1]$ regulates how fast a node moves towards the midpoint of its phase neighbors. Indeed, the boundaries of this interval will only be considered in exceptional cases within this work:



(a) The network of five nodes during the settling phase.



(b) The network of five nodes reached the stable state of perfect desynchrony.



(c) Detail of the timeline during the settling phase of the network of five nodes, and of node i in particular.

Figure 4.4: Snapshots of the progress of desynchronization for a network consisting of five sensor nodes. The circumference of the circle corresponds to the common period T.

- Setting $\alpha = 0$ means no movement at all. This is not desired in general, since it would simply disable this approach.
- Setting $\alpha = 1$ forces the node to always jump directly onto the current midpoint of its phase neighbors without any damping. This straight behavior could result in the emergence of new but unstable configurations (cf. [73] as well as Section 7.3).

In [50], Degesys et al. suggest $\alpha = 0.95$ as damping factor for single-hop topologies. However, further analysis of an optimal value for the damping factor α for multi-hop topologies is mandatory and can be found in Section 7.3. A snapshot of this settling phase is depicted in Figure 4.4(a), the corresponding timeline is shown in Figure 4.4(c).

Finally, after its current firing at time t_i node i is able to set its next (absolute) time of firing t_i^+ as

$$t_i^+ \overset{(4.6)}{=} t_i + T + \varphi_i\left(N_R(i), t_i\right) \tag{4.14a}$$

$$\overset{4.13}{=} t_i + T + \alpha \cdot \varepsilon_{t_i} \tag{4.14b}$$

$$= t_i + (1-\alpha) \cdot T + \alpha \cdot \left(\varepsilon_{t_i} + T\right). \tag{4.14c}$$

Based on Definition 4.3, the system has reached the stable state of perfect desynchrony, if there exists a point in time t such that for any future time of firing $t_i^{++} > t$ for each node i holds $\varepsilon_{t_i^{++}} = 0$. Figure 4.4(b) illustrates this stable state of perfect desynchrony.

Observation 4.11. Noteworthy, Eq. (4.14c) shows the algorithmic similarity of the midpoint approach to the *exponentially weighted moving average* (EWMA) filter[3] (cf. [155]), which smooths out short-term fluctuations but instead highlights long-term trends. In particular,

$$\overline{x}_k = \overline{\alpha} \cdot \overline{x}_{k-1} + (1 - \overline{\alpha}) \cdot x_k, \tag{4.15}$$

where the value of $\overline{x}_k$ is the filtered value of the currently observed value x_k combined with the recently filtered value $\overline{x}_{k-1}$. Again, the value of the filter constant $\overline{\alpha} \in [0,1]$, determines the degree of filtering.

4.3.1 Proof of Convergence

As already mentioned, the midpoint approach depends on simple calculations. This also simplifies the *proof of convergence*. However, we will show in Section 7.2 that the initial start up order of the nodes in a multi-hop topology has a significant impact on the temporal order of the nodes in general as well as on the temporal order of a node and its phase neighbors in particular. For this reason, the initial start up order also co-determines the convergence behavior of the whole system. Moreover, the configuration space of this midpoint approach is further expanded by the underlying topology and the used values for the protocol parameters, like the damping factor α. Therefore, we are able to prove the convergence of the midpoint approach just based on certain assumptions and simplifications.

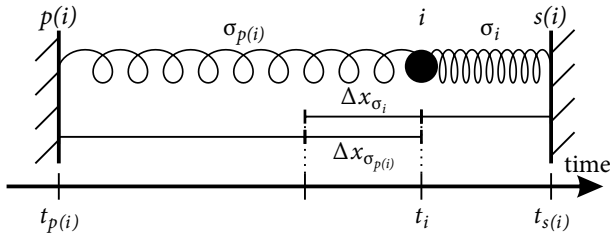
In contrast to the proof of convergence of the midpoint approach for a single-hop topology in [50], we do not try to convert the problem of desynchronization into the problem of graph coloring. Instead, we want to demonstrate the eligibility of other mapping approaches, like a proper physical model. Therefore, we utilize the physically inspired proof of an elastic resilience model as introduced by Mühlberger and Kolla in [128] to proof the convergence of the midpoint approach. Since the convergence of the midpoint approach was proven just for single-hop topologies, e.g., in [50], but still is missing for multi-hop topologies, we do focus on multi-hop topologies herein. Nevertheless, this proof also is compliant to single-hop topologies. In particular, we will focus on a specific multi-hop topology, namely a star topology $S_{|N|}$ consisting of the set N of nodes. For instance, the star topology S_5 for $|N| = 5$ nodes is depicted in Figure 2.3(b). Therefore, the midpoint approach has to be transferred into the corresponding elastic resilience model, first.

Convergence of the Midpoint Approach. To be able to proof the midpoint approach for the star topology $S_{|N|}$, we first have to make the following assumptions:

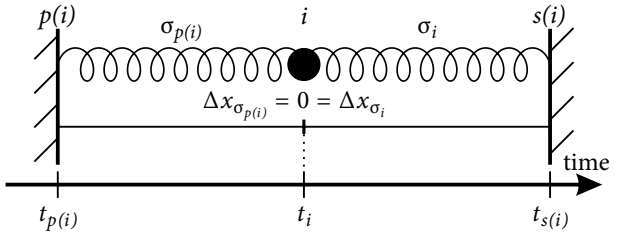
A1. One length unit equals one time unit, i.e., the period T can be mapped to a circle of circumference T.

A2. Due to the underlying star topology and according to Observation 4.7, each node requires its own, distinct time slot within period T for a collision-free communication.[4]

[3] The exponentially weighted moving average filter is identical to the *discrete first-order low-pass filter*.
[4] Noteworthy, Lemma 4.12 is not repealed thereby.



(a) During the settling phase: Coil spring $\sigma_{p(i)}$ is stretched by $\Delta x_{\sigma_{p(i)}}$, whereas coil spring σ_i is compressed by Δx_{σ_i} with $\Delta x_{\sigma_{p(i)}} = \Delta x_{\sigma_i}$.



(b) In perfect desynchrony: Both coil springs $\sigma_{p(i)}$ and σ_i are undeformed, i.e., $\Delta x_{\sigma_{p(i)}} = 0 = \Delta x_{\sigma_i}$.

Figure 4.5: Different stages of the elastic resilience model for our desynchronization approach. The black circle represents node i, and the walls represent $p(i)$ and $s(i)$.

A3. In accordance to Lemma 4.13, the period T has to be long enough to contain $|N|$ time slots for all $|N|$ nodes.

A4. The phase difference between a node $i \in N$ and any of its (phase) neighbors never exceeds T, i.e., Observation 4.2 applies here.

A5. We assume idealized conditions: All communication links are symmetrical, bidirectional, and reliable. Additionally, not any node will fail, and there is no clock drift.

A6. Since each node has to assign its own time slot, w.l.o.g. we are able to number all nodes consecutively, i.e., for a node's identifier i holds: $i \in \{0, \ldots, |N|-1\}$.

A7. According to Observation 4.9, for each node i holds: $i = p(s(i))$ and $i = s(p(i))$.

A8. To keep the proof manageable, we pick a certain node $i \in N$ for deeper analysis. In particular, although node i adjusts its next time of firing, its phase neighbors $s(i)$ and $p(i)$ are "frozen", i.e., they do not adjust their time of firing. Thus, $t^+_{s(i)} = t_{s(i)} + T$ as well as $t^+_{p(i)} = t_{p(i)} + T$ always holds.

Based on assumptions A1 to A8, we can define an elastic resilience model which complies to this network model (cf. Figure 4.5):

D1. Nodes are modeled as physical objects with identical mechanical characteristics, i.e., they are identical in space and mass.

D2. Each node $i \in N$ is linked to its successor $s(i)$ by a coil spring named σ_i. Consequently, the spring between node i and its predecessor $p(i)$ is named $\sigma_{p(i)}$.

D3. All coil springs of our elastic resilience model are identical in material and in construction. Especially, the (arbitrary) spring constant κ_{σ_i} as well as the undeformed length is equal for each coil spring σ_i.

D4. Since each node $i \in N$ is connected to its successor $s(i)$, and due to assumptions A2 and A7, all nodes can be arranged consecutively forming a closed loop. This means that all nodes and thus all springs are arranged on the circle of circumference T (cf. assumption A1) such that each node $i \in N$ is connected with its phase neighbors $p(i)$ and $s(i)$ via springs σ_i and $\sigma_{p(i)}$, respectively (cf. Figure 4.5).

D5. Noteworthy, due to the underlying star topology each phase neighbor is an element of the union $N_1(i) \cup N_2(i)$, i.e., a phase neighbor is one hop or two hops away from node $i \in N$.

D6. Furthermore, motion along the circle is frictionless for nodes as well as for springs.

D7. Moreover, the radius of this circle is constant.

D8. Finally, there is no external force at all.

Since a node receives information about its two-hop neighbors just by means of at least one of its one-hop neighbors (cf. Section 5.3), phase changes of two-hop neighbors are recognized within one period later. In combination with assumption A8, a phase neighbor which is two hops away can be treated as a one-hop neighbor with a delayed exchange of information then. Hence, it is legitimate to place them all along a single circle.

Besides, the springs are connected in series, trying to decrease their potential energy by returning to the equilibrium position. As soon as the nodes are distributed equidistantly along the circle, the resulting equilibrium of forces matches exactly with the stable state of (perfect) desynchrony. This means that after a settling phase, the stable state has reached the lowest potential energy of all springs accumulated. Therefore, since each node holds the largest possible temporal distance to each of its phase neighbors, it is sufficient to show that the midpoint approach also results in such a stable state.

Assuming $|N|$ nodes (and thus $|N|$ springs) with $0 \leq i \leq |N| - 1$ along a circle with circumference T as described above. The potential energy U_{σ_i} stored in spring σ_i then equals

$$U_{\sigma_i} = \frac{\kappa_{\sigma_i}}{2} \Delta x_{\sigma_i}^2, \tag{4.16}$$

where κ_{σ_i} denotes the spring constant and Δx_{σ_i} denotes the current displacement of spring σ_i (cf. textbooks on physics like [72, 102, 65, 181]). The potential energy U_N of all $|N|$ springs accumulates as

$$U_N = \sum_{i=0}^{|N|-1} \frac{\kappa_{\sigma_i}}{2} \Delta x_{\sigma_i}^2. \tag{4.17}$$

As mentioned above (cf. assumption D3), all springs have an arbitrary but identical spring constant. Therefore, for each coil spring σ_i we choose w.l.o.g. $\kappa_{\sigma_i} = 1$. Additionally, we are analyzing just a single node according to assumption A8. We refer w.l.o.g. to node $i \in N$ and thus obtain

$$U_N \overset{(4.17)}{=} \sum_{i=0}^{|N|-1} \frac{1}{2}\Delta x_{\sigma_i}^2 \tag{4.18a}$$

$$= \sum_{\substack{j=0,\\ j\neq i,\\ j\neq p(i)}}^{|N|-1} \frac{1}{2}\Delta x_{\sigma_j}^2 + \frac{1}{2}\Delta x_{\sigma_{p(i)}}^2 + \frac{1}{2}\Delta x_{\sigma_i}^2. \tag{4.18b}$$

Noteworthy, if the whole system would move (counter)clockwise along the circle without changing the relative distances amongst the nodes, the total energy of the elastic resilience model would remain constant. This is consistent with our definition of (non-perfect) desynchrony (cf. Definition 4.3).

Next, we have to transform the statements on the elastic resilience model made above into statements on a Wireless Sensor Network: For this reason, the temporal displacement of node $i \in N$ has to correlate with the displacement of the coil springs connected to this node as follows (cf. Figure 4.5(a)): With respect to the current time of firing t_i of node i, the value of its adjustment factor ε_{t_i} is equal to the value of the displacement Δx_{σ_i} of coil spring σ_i. Based on assumption A1, we are able to substitute ε_{t_i} for Δx_{σ_i}. Since all other nodes and especially the phase neighbors of node i will not move (cf. assumption A8), this adjustment factor ε_{t_i} also affects the displacement $\Delta x_{\sigma_{p(i)}}$ of coil spring $\sigma_{p(i)}$, since this spring is also connected to node i (cf. Figure 4.5). In consequence to assumption A7, we are also able to substitute ε_{t_i} for $\Delta x_{\sigma_{p(i)}}$. Utilizing assumption A4, Eq. (4.18b) further modifies to

$$U_N = \sum_{\substack{j=0,\\ j\neq i}}^{|N|-1} \frac{1}{2}\varepsilon_{t_j}^2 + \frac{1}{2}\varepsilon_{t_i}^2 + \frac{1}{2}\varepsilon_{t_i}^2 \tag{4.19a}$$

$$\overset{4.12b}{=} \sum_{\substack{j=0,\\ j\neq i}}^{|N|-1} \frac{1}{2}\varepsilon_{t_j}^2 + \left(\frac{t_{s(i)} + t_{p(i)}}{2} - t_i\right)^2. \tag{4.19b}$$

One important condition for proving the stable state of our system is that the difference in energy, when a single node $i \in N$ moves while all other nodes $j \in N$ with $j \neq i$ remain unchanged (cf. assumption A8), can be obtained by the partial derivative of the total potential energy U_N with respect to t_i, i.e.,

$$\frac{\partial}{\partial t_i} U_N = \frac{\partial}{\partial t_i} \left(\sum_{j=0, j\neq i}^{|N|-1} \frac{1}{2} \varepsilon_{t_j}^2 + \left(\frac{t_{s(i)} + t_{p(i)}}{2} - t_i \right)^2 \right) \tag{4.20a}$$

$$= \underbrace{\frac{\partial}{\partial t_i} \sum_{j=0, j\neq i}^{|N|-1} \frac{1}{2} \varepsilon_{t_j}^2}_{=0} + \frac{\partial}{\partial t_i} \left(\frac{t_{s(i)} + t_{p(i)}}{2} - t_i \right)^2 \tag{4.20b}$$

$$= \frac{\partial}{\partial t_i} \left(\frac{t_{s(i)} + t_{p(i)}}{2} - t_i \right)^2 \tag{4.20c}$$

$$= 2 \cdot \left(\frac{t_{s(i)} + t_{p(i)}}{2} - t_i \right) \cdot (-1) \tag{4.20d}$$

$$= 2 \cdot t_i - t_{s(i)} - t_{p(i)}. \tag{4.20e}$$

After some settling phase, there is no more change in energy, and the elastic resilience system enters a stable state (cf. Figure 4.5(b)). Thus, it is a necessary condition to finally have a minimum difference of energy, expressed by

$$\frac{\partial}{\partial t_i} U_N = 0. \tag{4.21}$$

Utilizing Eqs. (4.20e) and (4.21), we obtain

$$t_i = \frac{t_{s(i)} + t_{p(i)}}{2}. \tag{4.22}$$

In combination with the particular specification of the adjustment function (cf. Eq. (4.13)), Eq. (4.22) fully complies with our Definition 4.3 of (perfect) desynchrony. Furthermore, due to assumption A4, Eq. (4.22) is also conform to the (denormalized) adjustment factor in Eq. (4.12b). This conformity validates our substitution.

So far, we have showed that the elastic resilience system complies well with the midpoint approach. Furthermore, we characterized the stable state of that system. Since such a stable state exists, it will be attained eventually, if the change of the total potential energy of subsequent states decreases in a strictly monotonic way. This holds, until the stable state of desynchrony is reached actually. In the case of desynchrony, $\Delta U_N = 0$ holds. Thus, to demonstrate the emergence of this stable state, we have to show

$$\Delta U_N = U_N^+ - U_N < 0, \tag{4.23}$$

where U_N^+ denotes the potential energy of all $|N|$ springs after solely node $i \in N$ has changed its time of firing from t_i to t_i^+. This means that just the springs $\sigma_{p(i)}$ and σ_i are affected, since all other nodes $j \in N$ with $j \neq i$ remain unchanged (cf. assumption A8). According

to the midpoint approach in Section 4.3 and due to assumption A4, we use the following substitution

$$\begin{aligned}
\varepsilon_{t_i^+} &= \frac{t^+_{s(i)} + t^+_{p(i)}}{2} - t_i^+ && (4.24a)\\
&\overset{A8}{=} \frac{t_{s(i)} + T + t_{p(i)} + T}{2} - t_i^+ && (4.24b)\\
&\overset{4.14b}{=} T + \frac{t_{s(i)} + t_{p(i)}}{2} - \left(t_i + T + \alpha \cdot \varepsilon_{t_i}\right) && (4.24c)\\
&= \frac{t_{s(i)} + t_{p(i)}}{2} - t_i - \alpha \cdot \varepsilon_{t_i} && (4.24d)\\
&\overset{4.12}{=} \varepsilon_{t_i} - \alpha \cdot \varepsilon_{t_i} && (4.24e)\\
&= (1-\alpha) \cdot \varepsilon_{t_i} && (4.24f)
\end{aligned}$$

to prove Eq. (4.23) as

$$\begin{aligned}
\Delta U_N &= U_N^+ - U_N && (4.25a)\\
&= \varepsilon^2_{t_i^+} - \varepsilon^2_{t_i} && (4.25b)\\
&\overset{4.24f}{=} \left((1-\alpha) \cdot \varepsilon_{t_i}\right)^2 - \varepsilon^2_{t_i} && (4.25c)\\
&= (1-\alpha)^2 \cdot \varepsilon^2_{t_i} - \varepsilon^2_{t_i} && (4.25d)\\
&= \left((1-\alpha)^2 - 1\right) \cdot \varepsilon^2_{t_i}. && (4.25e)
\end{aligned}$$

Since we excluded $\alpha = 0$ as well as $\alpha = 1$ in Section 4.3, i.e., $\alpha \in (0,1)$ holds, we further evaluate Eq. (4.25e) as follows

$$\Delta U_N = \underbrace{\underbrace{\left((1-\alpha)^2 - 1\right)}_{<0} \cdot \underbrace{\varepsilon^2_{t_i}}_{\geq 0}}_{\leq 0}. \tag{4.26}$$

As long as the system is not in the stable state of perfect desynchrony (cf. Definition 4.3), $\varepsilon_{t_i} \neq 0$ and thus $\varepsilon^2_{t_i} > 0$ holds. Together with Eq. (4.26) follows

$$\Delta U_N < 0,$$

i.e., the total potential energy of subsequent states decreases in a strictly monotonic way as long as the system is not in the stable state of perfect desynchrony. Consequently, the convergence of the midpoint approach is proved. □

Indeed, according to Lemmas 4.7 to 4.13, the proof of convergence for multi-hop topologies without simplistic assumptions (cf. Items A1 to A8) is very hard – especially in combination with topology dynamics. A first impression of that difficulty is given in [49]. As stated in [170], it is impossible to obtain an analytical solution for most nonlinear systems in general.

Besides, [112] gives another evidence that no numerical proof for arbitrary multi-hop topologies exists. Moreover, the conclusions from Section 4.2 identify the respectable complexity when realizing the PCO framework for multi-hop topologies. For this reason, the proof of convergence is available at present just for single-hop topologies. Besides the easy-to-handle star topologies in [38] and in this section, a universal proof for arbitrary multi-hop topologies is still missing (cf. [34]).

Indeed, a more realistic model without some (or all) of the limiting assumptions Items A1 to A8 may never reach the stable state of desynchrony (cf. Eq. (4.7)). This could be caused by awkward start up scenarios or by too many erroneous nodes (cf. Section 7.2). However, we identified yet another problem which will be introduced in Section 5.9. This problem is inherent to the primitive of desynchronization and causes the system to reach an unstable but fluctuating state. Therefore, the midpoint approach has to be enhanced further (cf. Chapter 6).

4.3.2 Related Work

As already mentioned, the first implementation of the primitive of desynchronization as MAC protocol is based on this midpoint approach: In 2007, Degesys, Rose, Patel, and Nagpal first published DESYNC, a self-organizing TDMA protocol for fully-meshed WSNs [50]. Its straight forward implementation of the primitive of desynchronization from Section 3.4 allows another simple *proof of convergence* for complete networks: Since the exclusive assignment of disjoint time slots is similar to the problem of *graph coloring*, Patel, Degesys, and Nagpal tried to transform the DESYNC algorithm into such a linear dynamical system in [141]. Using Jacobian matrices, they proved that the stable state of perfect desynchrony equals the unique globally stable fixed point of the dynamics based on graph coloring. Indeed, this MAC protocol was intended for complete network graphs and thus operates just well for single-hop topologies. Since this restriction limits the amount of possible application scenarios, Degesys and Nagpal suggested an implementation for multi-hop topologies in [49]: In order to solve the emerging hidden terminal problem (cf. Definition 2.25), each node additionally requires knowledge about its two-hop neighborhood (cf. Section 5.3). However, each node is still aware of only its one-hop neighborhood. Furthermore, there was no realization of how to get the required knowledge about the nodes of the corresponding constraint graph in a decentralized but self-organizing network.

Taking advantage of the fact that the time span between any pair of successively transmitting nodes within a perfectly desynchronized and complete network always equals $T/|N|$ (cf. Lemma 4.5), Taechalertpaisarn et al. developed in [173] the orthodontics-inspired algorithm DESYNC-ORT to speed-up the process of desynchronization for single-hop topologies with just symmetrical links. Within such a complete network, each node $i \in N$ can autonomously determine $|N|$ (cf. Lemma 4.1). This information allows each node i to decide if it is already perfectly desynchronized according to Definition 4.3 (i.e., $\phi(t_i, t_{s(i)}) = T/|N| = \phi(t_{p(i)}, t_i)$), or if it still has to adjust its next time of firing according to the midpoint approach. Therefore, each node, which is already perfectly desynchronized (cf. Definition 4.3), simply keeps its phase, i.e., for each node $i \in N$ holds $t_i^+ = t_i + T$. With it, the impact of obsolete information is reduced. However, since the time span between successively transmitting

nodes may not equal $T/|N|$ in multi-hop topologies (cf. Lemma 4.11), this approach is just feasible for single-hop topologies and thus does not fulfill our requirements from Section 1.2.

In [159], Settawatcharawanit et al. adapted the midpoint approach to the Inter-Vehicular Communication (cf. Definition 2.5) of single-hop topologies. Hence, the so-called V-Desync protocol additionally has to meet the special requirements emerging in car-to-car communication. Therefore, Settawatcharawanit et al. aim for a contact time between passing nodes of just a few periods. Assuming 250 m as communication range and 120 km/h ≈ 33.3 m/s as maximum velocity for each mobile node, the period was set to $T \geq 1$ s for the simulations (cf. [159]). As the firing packets of concurrently transmitting vehicles would collide, such nodes will be not able to communicate with each other. Hence, each node $i \in N$ implementing the V-Desync protocol has to add a uniformly chosen random offset to its adjustment factor (cf. Eq. (4.11)). This probabilistic factor should reduce the possibility of colliding packets. Furthermore, each node could run at its own period length, i.e., periods could be multiples as well as non-multiples from each other. To support different period lengths, V-Desync forces each node to include the individual period within its firing packet to classify such "virtual" nodes. Each receiver with shorter period reserves the corresponding firing times for such "virtual" nodes. Indeed, this protocol is also just applicable for single-hop topologies.

The midpoint approach was also implemented successfully for periodic resource scheduling. In particular, Giusti, Murphy, and Picco developed in [73] an algorithm for the task of duty-cycling on single-hop topologies – especially for the decentralized scattering of wake-up times of sensor nodes. This algorithm is closely related to the midpoint approach presented above, since the relevant nodes are once more just the node's predecessor and its successor. Indeed, each node first listens to the wake up times of its neighbors within several so-called *calibration rounds*. Additionally, Giusti et al. also considered a solution for tree-based multi-hop networks: To avoid unstable behavior like fluctuations within such topologies, the adjustment of wake-up times is applied only with a given probability. The value of this probability parameter for each node depends on the length of the path from the root to this node, i.e., nodes close to the root should use higher probability values. With such a depth-related probability parameter, the system also converges faster. However, this multi-hop approach is not well-defined, i.e., pseudocode and algorithmic details (for instance about the formation of the tree and the information propagation) are missing. Moreover, there are no analyses about the impact of topology dynamics. In [140], Palopoli, Passerone, Murphy, Picco, and Giusti further optimized the scattering algorithm for multi-hop topologies: But instead of improving the algorithm in a self-organizing manner, they just integrated a central control acting contrary to our requirements.

4.4 The Local Max Degree Approach

Compared to the midpoint approach (cf. Section 4.3), nodes implementing the *local max degree* approach do not adjust their time of firing iteratively. Instead, each node first determines a sufficient number of time slots required for a collision-free communication within its neighborhood autonomously, since this number defines the corresponding length of each time slot. Indeed, the common period T as well as the *local max degree* D_i of a node $i \in N$ limit the minimum number of slots required for single-hop as well as for acyclic multi-hop topologies.

Definition 4.5 Local Max Degree, Global Max Degree. The *local max degree* D_i of a node $i \in N$ equals the maximum degree of its one-hop neighbors and the node itself:

$$D_i = \max\left\{d_j \colon j \in N_1(i) \cup \{i\}\right\}. \tag{4.27}$$

According to Definition 2.16, d_j denotes the degree of node j. Consequently, the *global max degree* D_N of a network consisting of the set N of nodes equals the maximum local max degree of all nodes of this network:

$$D_N = \max\{D_i \colon i \in N\}. \tag{4.28}$$

To obtain the local max degree, each node must collect information about the degree of its one-hop neighbors. Therefore, each node $i \in N$ has to broadcast its current degree d_i, i.e., the size of its currently known one-hop neighborhood (cf. Definition 2.16). With it, node i can determine its local max degree D_i, and hence, it can estimate a minimum number of time slots required for a collision-free communication. Afterwards, node i repeatedly tries to use one of these D_i slots for its own firings – as long as there is no more collision, i.e., as long as not any other node also had assigned the very same time slot.

Hence, the set of relevant nodes $N_R(i)$ of a node $i \in N$ here equals its one-hop neighborhood, i.e., $N_R(i) = N_1(i)$. The adjustment function $\varphi_i\left(N_R(i), t_i\right)$ just has to choose (for instance to randomize) one of the offered time slots. Finally, if a node $i \in N$ has successfully assigned a time slot, the next time of transmission results in a periodical transmission for this fixed time slot without further adjustments, i.e., $t_i^+ = t_i + T$ holds.

However, each node $i \in N$ first has to exchange information about its current degree d_i with all its one-hop neighbors to determine its local max degree D_i. This *exchange stage* may be lengthy due to collisions[5]. Depending on the currently implemented algorithm, the competitive *selection stage*, where each node has to select one of the available time slots uniquely, also takes a while – definitely as long as there are no more conflicts with nearby nodes for the very same time slot. Finally, both the lengthy exchange stage as well as the non-deterministic selection stage will be restarted in case of any topology dynamics. Therefore, the local max degree approach is neither very robust nor flexible (regarding topology dynamics).

4.4.1 Related Work

Motskin, Roughgarden, Skraba, and Guibas designed in [119] a desynchronization algorithm based on the local max degree approach for multi-hop topologies: Each node first divides the common period T into $2 \cdot (D_i + 1)$ slots, and next assigns one of these available slots randomly – as long as there are still collisions, i.e., as long as there are at least two nodes competing for the very same time slot. Indeed, Motskin et al. proved that their algorithm converges with high probability within $\mathcal{O}\left(D_N \cdot \log|N|\right)$ periods (cf. [119]). However, approximately half of the provided slots remain unassigned which wastes bandwidth and increases communication latency. Furthermore, each node requires knowledge about the global max degree D_N for this fast convergence. Therefore, each node's local max degree has to be propagated to each other node of the network (e.g., by *flooding*) causing high communication costs (especially in multi-hop topologies).

[5] The exchange stage just applies a contention-based back-off algorithm to access the shared communication medium since a more sophisticated access control scheme is unavailable at present (cf. [92]).

The M-Desync algorithm of Kang and Wong [92] for acyclic multi-hop topologies is also based on the local max degree D_i. However, M-Desync tries to maximize the slot utilization, i.e., to get along with the minimum number of required slots. Therefore, Kang and Wong proved for single-hop as well as acyclic multi-hop topologies that the minimum number of slots required per period for a collision-free communication within the communication range of a node $i \in N$ equals the local max degree D_i of this node plus 1: If each node knows its local max degree D_i, depending on the common period T, it can autonomously determine the maximum length of a time slot. As already mentioned, the real slot assignment occurs within the selection stage. To speed up this competitive process, the assignment of the $D_i + 1$ time slots is now prioritized according to the node's local max degree (cf. [92]). Nevertheless, the M-Desync algorithm is not applicable for cyclic multi-hop topologies (cf. [122]).

4.5 The Frog-Call Inspired Approach

This approach is also inspired by nature, since it is closely related to the mating calls of the male Japanese Tree Frog (*Hyla japonica*) which tries to attract female Japanese tree frogs in this way. Aihara, Kitahata, Yoshikawa, and Aihara developed in [3] a mathematical model, which utilizes the PCO framework from Section 3.2 (cf. also Section 3.4). However, in comparison to the approaches named above, a periodically croaking frog does not regulate its phase according to its relevant frogs. Instead, each frog changes its firing frequency to adjust its next time of "croaking" (i.e., firing).

According to its natural occurrence, where each frog relies on the firings of just nearby frogs, this approach was designed for single-hop topologies. Therefore, the set $N_R(i)$ of relevant nodes of a node $i \in N$ equals its set of one-hop neighbors, i.e., $N_R(i) = N_1(i)$. Similar to the algorithm of Werner-Allen, Tewari, Patel, Welsh, and Nagpal [192], a node does not react immediately every time it receives a firing. Instead, each node first accumulates the stimuli it is receiving within a single period. Next, it emits its own stimulus, and finally reacts to the received stimuli as a whole. As suggested by Aihara et al. in [3], the frog-call inspired approach utilizes a sinusoid as adjustment function. Using a sinusoid seems suitable for this reason, since this sort of function is as well periodic as the firings of each node. Besides, the amplitude of a sinusoid is always greater or equal 0 but can be monitored by an additional coefficient.

Indeed, for proper operation the frog-call inspired approach requires the network to be organized as rooted tree a priori (cf. [80]). This is feasible neither for networks with high topology dynamics nor for networks containing non idealistic and asymmetrical links. Furthermore, the computation of a sine is a challenging task – especially for sensor nodes which nowadays still provide just low computational power (cf. Definition 2.6). In this regard, there may exist more efficient algorithms to compute complex mathematical functions, for instance for the square root function in [21]. Using a *lookup table* (LUT) containing precalculated function values is always of just limited length and thus may be neither sufficient nor suitable. Besides, the memory of sensor nodes is also limited (cf. Definition 2.6). Furthermore, the frog-call inspired approach is only applicable for single-hop topologies, at least an extension for multi-hop topologies is still missing.

4.5.1 Related Work

As already mentioned, Aihara et al. suggest for their mathematical model in [3] a sinusoid as adjustment function: The sine of the phase shift between node $i \in N$ and its one-hop neighbor $j \in N_1(i)$ is multiplied by the corresponding *coupling coefficient* $\kappa_{ij} > 0$. This coupling coefficient represents the strength of the (pairwise) coupling between both nodes. With it, the adjustment function for a node $i \in N$ at its firing time t_i equals

$$\varphi_i\left(N_1(i), t_i\right) = -\frac{1}{|N_1(i)|} \sum_{j \in N_1(i)} \kappa_{ij} \cdot \sin\left(\phi(t_i, t_j)\right). \tag{4.29}$$

However, due to the symmetric sine function, the sum in Eq. (4.29) divides the nodes of the complete network (consisting of more than one node) into groups of two or three nodes (cf. [132, 133]). Therefore, this implementation just works well for single-hop topologies consisting of at most three nodes.

Within his master thesis [131], Mutazono tried to apply the frog-call inspired approach for single-hop topologies consisting of more than three nodes. For this purpose, a receiving node $i \in N$ first weights every firing from a one-hop neighbor $j \in N_1(i)$ according to the *phase distance* $\delta_{i,j} \in [0, T)$, which mainly depends on the absolute value of the corresponding phase shift (cf. Eq. (4.2)):

$$\delta_{i,j} = \min\left\{\left|\phi(t_i, t_j)\right|, T - \left|\phi(t_i, t_j)\right|\right\}. \tag{4.30}$$

Next, for the sake of simplicity, the coupling coefficient is equalized to a common value κ regardless of the applied pair of nodes. Finally, the adjustment function for a node $i \in N$ at time t_i from Eq. (4.29) is slightly modified to

$$\varphi_i\left(N_1(i), t_i\right) = \sum_{j \in N_1(i)} \kappa \cdot \sin\left(\phi(t_i, t_j)\right) \cdot \mathrm{e}^{-\delta_{i,j}}. \tag{4.31}$$

Due to the weighted adjustment function, even large single-hop networks are able to desynchronize well (cf. [132, 133]). Notably, Mutazono uses the term *anti-phase synchronization* as a synonym for *desynchronization* (cf. Observation 2.13).

Hernández and Blum further improved the frog-call inspired approach for single-hop topologies (cf. [80, 81]). They added a so-called *relevance parameter*, which depends on the number of received firings during the current period: Messages from nodes, which were influenced by many other nodes, should have less weight. Therefore, messages from nodes being influenced just marginally by other nodes get a higher weight. This adaption results in a faster convergence requiring less communication rounds. However, for proper operation, the relevance parameter also has to be transmitted in every single firing message.

4.6 The Artificial Force Field Approach

This approach originates from the robotic pattern formation: Mobile robots without global knowledge but just a limited visibility range have to evenly distribute themselves to perform a circle pattern. Forming a regular circle pattern in a self-organizing manner is a quite complex

task. Therefore, Boonpinon and Sudsang successfully installed in [30] an *artificial force field* as a helpful abstraction for the velocity adaptation of each robot.

Indeed, this self-organizing technique of an artificial force field in a spatial domain can also be implemented inside the temporal domain, namely to establish a TDMA protocol for Wireless Sensor Networks: The firing times of sensor nodes, which should be spread out temporarily equidistant over the common period T, are substituted for the nicely spaced mobile robots. The artificial force field then is equivalent to the length of a circle's circumference T, i.e., nodes within the same artificial force field are interfering with each other.

A single erroneous time of firing of a node $j \in N_R(i)$ impacts the next time of firing of node $i \in N$ more significantly when the set $N_R(i)$ of relevant nodes is small[6]. For this reason, the artificial force field approach utilizes a large set of relevant nodes to smooth out erroneous data about relevant nodes: The set $N_R(i)$ of relevant nodes of a node i equals the union of the set $N_1(i)$ of its one-hop neighbors and the set $N_2(i)$ of its two-hop neighbors, i.e., $N_R(i) = N_1(i) \cup N_2(i)$. Nevertheless, the firing time of a node i is "pushed away" from the firing time of each of its relevant nodes $j \in N_R(i)$ according to the artificial *repelling force* $f_{i,j}$. Indeed, this approach demands that each node obtains information about all its relevant nodes at every period. This demand hinders the integration of techniques for energy saving, since the permanent demand for current information causes high communication costs.

The smaller the temporal distance between the firings of two interacting nodes i and j, the higher the magnitude of this repelling force should be. This effect is described by the *phase difference* $\Delta_{i,j} \in \left[-\frac{T}{2}, \frac{T}{2}\right]$ between both nodes i and j, which depends on the phase distance $\delta_{i,j}$ from Eq. (4.30):

$$\Delta_{i,j} = \begin{cases} \delta_{i,j} - T & \text{if } \delta_{i,j} > \frac{T}{2} \\ \delta_{i,j} & \text{otherwise} \end{cases}. \tag{4.32}$$

With it, the magnitude of the repelling force $f_{i,j}$ between both nodes i and j equals

$$f_{i,j} = -\frac{T}{\Delta_{i,j}}. \tag{4.33}$$

The critical case $\Delta_{i,j} = 0$ represents nodes i and j both firing at the (same) time, i.e., $t_i = t_j \bmod T$ and vice versa. However, this case is very unusual in single-hop topologies, where it could be ignored then. Indeed, it could arise in multi-hop topologies and has to be handled explicitly therefore. Furthermore, node j does not repel node i, i.e., $f_{i,j} = 0$, if both nodes i and j are balanced, i.e., if $\Delta_{i,j} \in \left\{-\frac{T}{2}, \frac{T}{2}\right\}$. This case has to be taken into account explicitly as well. Since the phase difference of a balanced pair of nodes is ignored in the related work, the open interval for the phase difference is used therefore. The sum of all forces perceived by node i during its current period results in the *total force* F_i, i.e.,

$$F_i = \sum_{j \in N_R(i)} f_{i,j}. \tag{4.34}$$

Finally, the adjustment function $\varphi_i\left(N_R(i), t_i\right)$ of a node i at its firing time t_i equals the total force F_i weighted by the *jump size parameter* $\alpha_{\mathrm{AF}} \geq 0$, i.e.,

$$\varphi_i\left(N_R(i), t_i\right) = \alpha_{\mathrm{AF}} \cdot F_i. \tag{4.35}$$

[6] Comparable to the midpoint approach, which utilizes just the phase neighbors.

This damping coefficient α_{AF} is similar to the jump size parameter α from Section 4.3, since it regulates the convergence behavior of the system: The system may converge just very slowly, if the value of α_{AF} is too small, whereas it may overshoot and even may not converge at all, if the value of α_{AF} is too large. Therefore, setting $\alpha_{\mathrm{AF}} = 0$ is allowed (cf. [38]) but is not very useful – as for α in Section 4.3.

However, a sufficient value for α was identified just empirically (cf. [50]), whereas a sufficient value for α_{AF} was determined analytically this time: The phase difference between two nodes tend to be smaller in a dense network than in a sparse network. As a result, a node in a sparse network may have to make a bigger adjustment to its optimum next time of firing. The corresponding coefficient α_{AF} should take these circumstances into account. Therefore, it has to be inversely proportional to the power of the number of relevant nodes.

Since the total force corresponds to the net force of a mechanical system, a node $i \in N$ is in an equilibrium state, if its total force is equal to 0 (cf. *mechanical equilibrium*). Due to Eq. (4.35), the whole system has reached the stable state of (perfect) desynchrony, if each node $i \in N$ of the network is in an equilibrium state, i.e., for each $i \in N$ holds: $F_i = 0$.

However, the determination of the next time of firing involves high computational costs, since there are several divisions required to calculate the total force from the received firing packets within each period. Moreover, the communication costs are also high, as information about all relevant nodes is required at every period. Mainly because of the low computational power of sensor nodes nowadays (cf. Definition 2.6), the artificial force field approach is of just limited suitability regarding our purposes.

4.6.1 Related Work

DWARF [39] is the first MAC protocol for single-hop topologies which implements the artificial force field approach. The main contribution of this protocol is to reduce the impact of erroneous information at each node. For this purpose, the set of relevant nodes of a node $i \in N$ is equal to its set of one-hop neighbors, i.e., $N_R(i) = N_1(i)$. This allows a fast convergence as well as a high robustness to topology dynamics, e.g., nodes joining or leaving the network. In [39], the proper value for the damping factor has been determined by experiments as

$$\alpha_{\mathrm{AF}} = 38.597 \cdot |N_R(i)|^{-1.874} \cdot \frac{T}{1000}. \tag{4.36}$$

This computationally demanding coefficient makes the calculation of the next time of firing even more expensive: A precalculated lookup table for certain values of α_{AF} could save this costly runtime computation. However, similar to the frog call approach (cf. Section 4.5), such a lookup table always contains just a limited number of values and requires memory, which is still restricted at sensor nodes nowadays (cf. Definition 2.6).

The M-DWARF protocol by Choochaisri [38] is the multi-hop extension of the DWARF protocol named above. The multi-hop variant has to solve the hidden terminal problem which is inherent to multi-hop topologies (cf. Definition 2.25). For this purpose, Choochaisri installed our *phase shift propagation*, which was proposed first by Mühlberger and Kolla in [128] and will be explained in detail in Section 5.3. Furthermore, due to Lemma 4.12, there could be two (or more) relevant nodes of a node $i \in N$ which are firing at the same time, e.g., for a node $j \in N_1(i)$ and for a node $k \in N_2(i)$ let $t_j = t_k$. Corresponding to the artificial force field

approach, the identical forces $f_{i,j} = f_{i,k}$ would both repel node i. However, this treatment does not agree to the primitive of desynchronization, at least one of these forces has to be absorbed. Therefore, Choochaisri additionally developed the *force absorption mechanism* in [38], which absorbs the overwhelming force from at least two nodes as follows: At every period, each node $i \in N$ sorts the received times of firing of its relevant nodes by the ascending absolute value of the corresponding phase difference, resulting

- in a sorted list of successive neighbors $L_S(i) = \{s(i) = s(i)_1, s(i)_2, s(i)_3, \ldots\}$, where $s(i)_x$ denotes the x-th next neighbor, as well as
- in a sorted list of preceding neighbors $L_P(i) = \{p(i) = p(i)_1, p(i)_2, p(i)_3, \ldots\}$, where $p(i)_x$ denotes the x-th previous neighbor.

With it, the *absorbed force* $f'_{i,j}$ from a successive neighbor $j = s(i)_x$ to a node i equals

$$f'_{i,j} = \begin{cases} f_{i,j} & \text{if node } j \text{ is phase neighbor of node } i, \text{ i.e., } x = 1 \text{ holds} \\ f_{i,s(i)_{x+1}} - f_{i,j} & \text{otherwise} \end{cases}. \tag{4.37}$$

This applies analogously to a preceding neighbor $j = p(i)_x$.

Choochaisri also tried to prove the convergence of its M-Dwarf algorithm within his dissertation [38] by maintaining the damping factor α_{AF} from the single-hop variant (cf. Eq. (4.36)). In particular, Choochaisri was able to demonstrate that the M-Dwarf algorithm keeps a star topology with an even number of nodes within the stable state of perfect desynchrony – even under small perturbation[7] (cf. [38]). However, the firing times of the nodes of a star topology at perfect desynchrony are distributed as equidistantly as at a single-hop topology consisting of the same number of nodes (cf. Observation 4.8).

[7] The magnitude of a tolerable perturbation was not further explained.

Chapter 5

The extended-Desync Protocol

Abstract

This chapter describes the EXTENDED-DESYNC protocol, which *extends* the idea of the midpoint approach from single-hop topologies to multi-hop topologies (cf. Section 5.1). In consequence to Observation 2.8, we thus have to cope with the hidden terminal problem: Section 5.2 presents two potential approaches solving this problem without a central coordinator. Since these solutions are still not sufficient for our demands, we developed the *phase shift propagation* (PSP) approach, which is specified in Section 5.3. The timing issues of the phase shift propagation are further examined in Section 5.4, whereas Section 5.5 analyzes the proper information to be exchanged between neighboring nodes. On this basis, the packing of this information is discussed in Section 5.6. The intended frame structure including application data is presented in Section 5.7. In Section 5.8, we describe practical issues originating from real-world conditions to improve our EXTENDED-DESYNC protocol. Finally, Section 5.9 summarizes the properties of our EXTENDED-DESYNC protocol but also leads us to another problem which is strongly related to the primitive of desynchronization: the *stale information problem*, which will be addressed in Chapter 6.

5.1 Motivation

In Chapter 4, we discussed various approaches together with relevant implementations of the primitive of desynchronization as self-organizing MAC protocol for WSNs. Apart from the M-DWARF protocol, which is based on the artificial force field approach (cf. Section 4.6), the usability of the self-organizing protocols presented in Chapter 4 is strictly limited to single-hop topologies, or at its best, to acyclic multi-hop topologies (cf. Table 4.1).

Since one of our objectives is to develop a robust but self-organizing MAC protocol for arbitrary multi-hop topologies (cf. Section 1.2), the artificial force field approach looks quite promising (cf. Section 4.6). However, this approach as MAC protocol for multi-hop WSNs (cf. Section 4.6.1) not only causes high communication costs but it also requires high computational efforts due to the following reasons:

- Based on the information of all firing packets which have been received within a single period, node $i \in N$ has to perform several time-consuming divisions to calculate the total force F_i from the repelling forces of all its one-hop neighbors according to Eqs. (4.33) and (4.34).
- For this computation node i requires up-to-date information about its whole one-hop as well as its two-hop neighborhood in every period. This causes high communication costs.

- Moreover, the total force F_i is multiplied by the extensive damping factor α_{AF}, which also causes high computational efforts (cf. Eq. (4.36)).
- In fact, a precalculated lookup table for values of α_{AF} could save costly runtime computations. However, such a lookup table is hardly feasible: Due to the restricted memory at sensor nodes nowadays (cf. Definition 2.6 as well as Section 2.4), a lookup table can contain just a limited and insufficient number of such values.

In consequence, we prefer the midpoint approach (cf. Section 4.3) over the artificial force field approach as basis for our self-organizing MAC protocol due to the following reasons:

- In general, the amount of relevant nodes $N_R(i)$ of a node $i \in N$ in the implementation of the midpoint approach is less than (or equal to) the amount of relevant nodes in the implementation of any other approach from Chapter 4 (cf. Table 4.1).
- The smaller the set of relevant nodes, the less administration effort is necessary. This even may save communication costs.
- Due to its simple algorithm, the computation of the midpoint approach (cf. Eq. (4.13)) consumes less time and less memory (on sensor node hardware) than the computation of other approaches from Chapter 4 and of the artificial force field approach in particular.

Based on the midpoint approach from Section 4.3, we developed the EXTENDED-DESYNC protocol as lightweight and self-organizing MAC protocol for arbitrary multi-hop topologies in WSNs. Nevertheless, the main problem of an implementation for multi-hop topologies is the hidden terminal problem (cf. Definition 2.25): This problem may cause undesired packet collisions. Since this problem is inherent to multi-hop topologies, solutions like the RTS/CTS handshaking do exist. However, we will argue in the next Section 5.2, why the available solutions do not meet our demands. Furthermore, for any decentralized and self-organizing realization of the midpoint approach for multi-hop topologies, each node has to gain knowledge about its constraint graph (cf. Definition 2.28). Hence, we will use this fact in Section 5.3 to cope with the hidden terminal problem.

5.2 The Hidden Terminal Problem Revised

As already indicated, a feasible extension of the midpoint approach (cf. Section 4.3) for multi-hop topologies has to address the hidden terminal problem (cf. Definition 2.25) as well as the exposed terminal problem (cf. Definition 2.26). Since our protocol will only use broadcasts (within this work), it is sufficient to just concentrate on the hidden terminal problem. As a result of Observation 2.9, the exposed terminal problem is not relevant within this work and will not be taken into consideration furthermore.

In this section, we first illustrate two potential approaches which solve the hidden terminal problem in (acyclic) multi-hop topologies without a central coordinator, namely *Local Max Degree* in Section 5.2.1 and *RTS/CTS handshaking* in Section 5.2.2. However, neither approach is sufficient for our purposes (cf. Section 1.2):

- We have to solve the hidden terminal problem for arbitrary multi-hop topologies.

- Consequently, we do not assume bidirectional links, i.e., we have to support asymmetrical and even unidirectional links (cf. Definition 2.9).
- We expect topology dynamics, i.e., changes in the underlying topology are likely and have to be considered.

Therefore, we developed our phase shift propagation approach, which will be introduced in Section 5.3.

5.2.1 The Local Max Degree

As mentioned in Section 4.4, the local max degree is equal to the provable minimum number of time slots required for a collision free but periodic communication within acyclic multi-hop topologies. That means, if a node has knowledge about its local max degree, it eventually is able to assign one of this minimum number of potential time slots. However, this method is not sufficient for our purposes. In particular, it suffers from a couple of drawbacks:

- To determine its own local max degree, each node has to exchange information about its current degree (cf. Definition 2.16) at first. This procedure could take a long time due to emerging collisions.[1]
- Next, each node selects one of the available time slots. This process is time consuming and lasts until each node has assigned one unique time slot. Hence, this process is quite competitive, and so collisions are very likely.
- Moreover, these long lasting stages, namely the exchange of degree information and the selection of a unique slot, have to be restarted after any topology change.
- Finally, the local max degree approach works just well for acyclic multi-hop topologies. However, our protocol has to operate in arbitrary and thus cyclic multi-hop topologies to fulfill our requirements from Section 1.2.

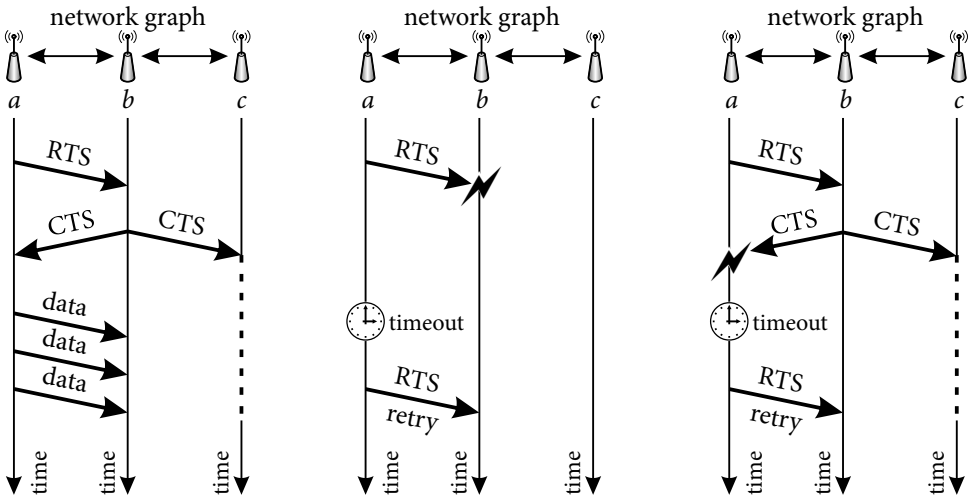
In summary, the local max degree method is just capable of acyclic multi-hop topologies. Additionally, it is neither robust nor flexible to topology dynamics. Therefore, to solve the hidden terminal problem, the local max degree approach does not meet our demands (cf. Section 1.2).

5.2.2 The RTS/CTS Handshake

The *RTS/CTS handshake* protocol (cf. [94, 86]) is another well-known approach to solve the hidden terminal problem in multi-hop topologies. This handshaking protocol is suitable especially for contention-based CSMA protocols.

To describe the RTS/CTS handshaking, we refer to a sample scenario with the linear topology L_3 of three nodes, namely node a, node b, and node c. As depicted in Figure 5.1, node a is not able to directly communicate with node c and vice versa:

[1] Please note that a reasonable MAC protocol, which shall reduce such collisions, is missing at this stage.



(a) Successful RTS/CTS handshake (b) No reception of RTS (c) No reception of CTS

Figure 5.1: Sample scenarios of the RTS/CTS handshaking to solve the hidden terminal problem.

- Assuming, node *a* wants to transmit data to its neighbor node *b*. Hence, node *a* initially has to broadcast a short *request-to-send* (RTS) to the desired receiver node *b*.
- If node *b* receives this RTS from node *a* correctly, i.e., before a certain timeout and without any collision (cf. Figure 5.1(b)), node *b* in return responds a short *clear-to-send* (CTS) which allows the requesting node *a* to transmit data for an appointed period of time (cf. Figure 5.1(a)).
- The RTS/CTS handshake is successful, if node *a* correctly (i.e., before a certain timeout and without any interference) receives the responding CTS from node *b*. As a result, merely node *a* is allowed to allocate the designated channel and to transmit its data to node *b* within the announced period of time (cf. Figure 5.1(a)).
- Otherwise, if node *a* did not receive the corresponding CTS to its former RTS, node *a* has to retransmit its request-to-send, for instance after a certain (and maybe mutable) *back-off* time (cf. Figure 5.1(b)).
- Since request-to-send as well as clear-to-send are transmitted as broadcasts, nearby nodes (here: node *c*) are able to register the announced data transmission. As a consequence, each overhearing node (cf. node *c* in Figure 5.1) will be silent (depicted as dotted line in Figure 5.1) during the requested data transfer. The objective is to not interfere the arranged communication (cf. Figure 5.1(a)) – even when the requesting node *a* did not receive the CTS correctly (cf. Figure 5.1(c)).

However, any implementation of the primitive of desynchronization as MAC protocol for WSNs follows a self-organizing manner. These implementations all result in a schedule-based TDMA protocol with periodic firings (cf. Chapter 4). This is in contrast to a contention-based CSMA protocol with maybe arbitrary transmission times but explicit requests. Nevertheless, the RTS/CTS handshake protocol explicitly *requests to send*. Therefore, it is incompatible with our basic idea and with the underlying algorithm for a self-organizing MAC protocol.

Besides, the RTS/CTS protocol seems to be not robust against topology dynamics. In addition, the RTS/CTS handshaking explicitly relies on bidirectional links to solve the hidden terminal problem. Hence, this approach does not meet our demands from above. For this reason, we developed the *phase shift propagation* (PSP), which is described in the following section.

5.3 Phase Shift Propagation

Both approaches presented in Section 5.2 do solve the hidden terminal problem, but are not sufficient for our purposes: The local max degree is not universally applicable, since it is just feasible for acyclic multi-hop topologies. Whereas, the RTS/CTS handshake is primarily designed for contention-based MAC protocols with arbitrary transmission times and on-demand communication. Thus, it does not comply with the primitive of desynchronization.

Nevertheless, the local max degree approach does not rely on a priori knowledge nor on a fixed schedule. Instead, it collects information about its neighborhood to create a competitive assignment of time slots. Despite the maybe long lasting competitive assignment procedure, such a collecting method could be helpful to solve the hidden terminal problem in a self-organizing manner. Therefore, we developed the *phase shift propagation* (PSP) approach.

After a short description of our basic idea in Section 5.3.1, we will proof in Section 5.3.2 that the collective propagation of information about one-hop neighbors enables each node to generate its constraint graph (cf. Definition 2.28) autonomously. In Section 5.3.3, we will introduce a *timed constraint graph*, i.e., we will discuss the preparation and integration of timing information to perform the necessary calculations according to the midpoint approach from Section 4.3.

5.3.1 Basic Idea

As shown in [49], information about the constraint graph (cf. Definition 2.28) is one helpful formalization to solve the hidden terminal problem for multi-hop topologies but to remain consistent with the primitive of desynchronization. Therefore, we developed the lightweight and universal applicable *phase shift propagation* (PSP), which enables each node to gather timing information about the nodes of its constraint graph autonomously. Such a method was first described in [68][2] and enhanced with respect to the primitive of desynchronization in multi-hop topologies by Mühlberger and Kolla in [128]. Apart from the EXTENDED-DESYNC protocol (cf. Chapter 5) and the EXTENDED-DESYNC$^+$ protocol (cf. Chapter 6), respectively, this method is also implemented in the M-DWARF protocol by Choochaisri (cf. [38] and Section 4.6) as well as in the protocol of Buranapanichkit in [33].

[2] Diploma thesis conducted in conjunction with this work.

The basic idea of the phase shift propagation approach is that each node repetitively propagates timing information about its set of one-hop neighbors. Ideally, the periodic firing packets are used for this purpose. As a consequence, each receiving node is able to further gain (temporal) knowledge about its two-hop neighbors.

5.3.2 Constraint Graph Creation

The first challenging problem for each node, which realizes the phase shift propagation approach, is the autonomous creation of its constraint graph: Due to the self-organizing manner, just locally available information can be used for this task, remote information has to be made local first. In fact, information about the node's one-hop neighborhood can be easily acquired in terms of incoming firing packets. Regarding the information about the node's two-hop neighborhood, each node depends on the collaboration of its neighbors:

Lemma 5.1 Constraint Graph Creation. *If each node $j \in N$ transmits information (i.e., the identifiers) about its one-hop neighborhood $N_1(j)$ within its firing packets, each receiver $i \in N$ is able to establish its constraint graph $\vec{G}_C(i)$ autonomously after a finite number of firing packets.*

Proof. Let each node $j \in N$ transmit its whole set $N_1(j)$ of one-hop neighbors within its firing packet. Since there is just a finite number of one-hop neighbors, the length of a firing packet is finite as well as the number of firing packets which have to be received. Further, let node $i \in N$ receive this firing packet from node j. In compliance with Definitions 2.15 and 2.16, $j \in N_1(i)$ holds. Hence, node i is able to create the graph $\vec{G}_{\mathcal{C}}(i) = \big(N_{\mathcal{C}}(i), E_{\mathcal{C}}(i)\big)$ as follows: Since node i received the firing packet from node j, the relations $i \in N_{\mathcal{C}}(i)$, $j \in N_{\mathcal{C}}(i)$, and $(j,i) \in E_{\mathcal{C}}(i)$ hold. Furthermore, since the firing packet of node j includes its one-hop neighborhood $N_1(j)$, and since this one-hop neighborhood contains all nodes previously received by node j (cf. Definition 2.16), the receiving node i is also able

- to add each node $k \in N_1(j)$ to its set $N_{\mathcal{C}}(i)$ of nodes, and
- to add each link (k,j) with $k \in N_1(j) \subseteq N$ to its set $E_{\mathcal{C}}(i)$ of links.

Because each node of the network broadcasts its one-hop neighborhood in this way, this is especially true for each one-hop neighbor $j \in N_1(i)$ of node i. Again, there is just a finite number of one-hop neighbors. Thus, after a finite number of firing packets for the finite set $N_{\mathcal{C}}(i)$ of node i holds

$$N_{\mathcal{C}}(i) = \{i\} \cup N_1(i) \cup \big\{k \in N_1(j) : \exists j \in N_1(i)\big\}, \tag{5.1}$$

and for the finite set $E_{\mathcal{C}}(i)$ of node i holds

$$E_{\mathcal{C}}(i) = \big\{(j,i) \in E : j \in N_1(i)\big\} \cup \big\{(k,j) \in E : \big(\exists j \in N_1(i) : k \in N_1(j)\big)\big\} \tag{5.2a}$$

$$\overset{(2.5)}{=} E_C(i). \tag{5.2b}$$

Since $N_{\mathcal{C}}(i)$ contains $\{i\}$ as well as $N_1(i)$ anyway, both sets can be excluded from the last subset in Eq. (5.1). As a result, we get

$$N_{\mathcal{C}}(i) = \{i\} \cup N_1(i) \cup \big\{k \in N_1(j) \setminus \{N_1(i) \cup \{i\}\} : j \in N_1(i)\big\} \tag{5.3a}$$

$$\overset{(2.4)}{=} N_C(i). \tag{5.3b}$$

Therefore, $N_{\mathcal{C}}(i) = N_C(i)$ and $E_{\mathcal{C}}(i) = E_C(i)$ are the corresponding sets of the graph $\vec{G}_{\mathcal{C}}(i)$, which was just created by node i autonomously. This graph is equal to the constraint graph $\vec{G}_C(i)$ of node i according to Definition 2.28, i.e., $\vec{G}_{\mathcal{C}}(i) = \vec{G}_C(i)$ holds. □

So far, each node is able to determine *who* (i.e., which node) is involved in *which way* (i.e., by which edge) within its constraint graph. However, for a proper operation of our self-organizing MAC protocol this information is not sufficient. Additionally, each node must be able to estimate *at which time* the (two-hop) neighbors of its constraint graph are firing to solve the hidden terminal problem in conjunction with the primitive of desynchronization. The integration of timing information results in a *Timed Constraint Graph* and will be introduced in the next section.

5.3.3 Timed Constraint Graph

In principle, it seems to be a simple task for a sensor node to acquire timing information about its one-hop and its two-hop neighbors: According to Definition 2.6, each sensor node operates a local clock. This local clock can be used to timestamp internal as well as external events. Consequently, each node is able to timestamp incoming firing packets (with more or less accuracy). Therefore, each node is able to achieve timing information about its one-hop neighbors autonomously, e.g., by timestamping incoming firing packets. To support other nodes in gathering timing information about their two-hop neighbors, each node has to

- record the reception time of any incoming firing packet from all its one-hop neighbors, and finally
- forward a collection of this information within its own firing packets.

As a result, any receiving node shall be able to calculate the time of firings of its two-hop neighbors – similar to Lemma 5.1 (cf. also Section 5.5). Nevertheless, the following issues have to be resolved therefore:

I1. The timestamping, i.e., how may a node obtain accurate and precise timestamps of incoming firing packets.

I2. The proper amount of information, i.e., which set of (timing) information is essential to be shared and how frequently has this information to be provided.

We will analyze issue I1 about timestamping in Section 5.4 and issue I2 about information quantity in Section 5.5, respectively.

5.4 Timing Issues

The functionality as well as the efficiency of our self-organizing MAC protocol EXTENDED-DESYNC strongly depends on the availability of accurate and precise timing information. Thus, it is important to get accurate timestamps of consistently high quality. Indeed, any implementation of the sketchy approach from Section 5.3.3 will face issues about *communication delays*, which will be described in Section 5.4.1, as well as issues about the process of accurate and precise *timestamping*, which will be described in Section 5.4.2.

Delay Type	Send	Medium Access	Propagation	Transmission / Reception	Receive
Caused by	OS & driver	MAC protocol	distance	data length & data rate	OS & driver
Namely (in testbeds)	SmartOS / SmartNET	extended-Desync / extended-Desync$^+$	< 1 km	≤ 64 B (100 kbps)	SmartOS / SmartNET
Magnitude	ms	µs	ns	ms	ms
Negligible	No	Yes	Yes	No	No
Predictability	Bad	Good	Good	Good	Bad

Table 5.1: Survey of different communication delays during a packet transmission.

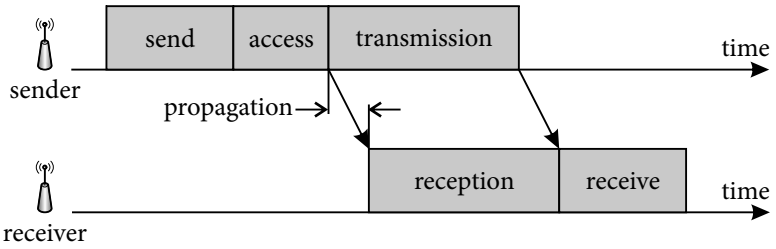


Figure 5.2: Different communication delays during a packet transmission.

5.4.1 Communication Delays

Our self-organizing MAC protocol neither relies on explicit time synchronization (cf. Section 1.2) nor is it introduced to operate as time synchronization protocol for sensor nodes (cf. Definition 2.34). Nevertheless, the necessary exchange of timing information to cope with multi-hop topologies is carried out wirelessly, and thus will be subject to comparable requirements as mature time synchronization protocols (cf. Section 8.2.1). Besides, any other wireless transmission of data also has to operate under these conditions. Although, some of these conditions may be ignored for specific contention-based applications and protocols (cf. Definition 2.31). Anyhow, the midpoint approach does rely on time. Consequently, these delays have to be considered within our extended-Desync protocol and – in particular – within our phase shift propagation approach from Section 5.3 as well.

The communication delays of the wireless communication process have been first analyzed by Kopetz and Ochsenreiter in [100]. Their classification has been extended by Maróti et al. in [111] (cf. also [48, 158, 70]). Since these delays also occur during the exchange of (timing) information, we will analyze the impact of these delays on our real-world testbed and the implementation of our extended-Desync protocol, respectively. A scheme of the sending/receiving process of two distinct nodes is depicted in Figure 5.2. To get a first impression about the impact of each condition on our real-world testbeds, Table 5.1 gives a short overview of these potential delays:

Send Delay The creation of a firing packet as well as its transfer to the communication interface takes time. For instance and representative for many sensor nodes, the microcontroller of a SNoW^5 node is connected to the radio unit via SPI (cf. Section 2.4). The

time between the send command of the application and the packet transfer to the radio unit is non-deterministic in general, and mainly depends on the operating system, the network driver, and the current workload of the involved hardware resources, like microcontroller, radio unit, and bus interface. According to the midpoint approach (cf. Section 4.3) and due to the underlying primitive of desynchronization, each node estimates its next time of firing in advance. As a consequence, the (start of the) transmission of the firing packet has to occur very closely to this specific time. Due to the run-to-completion scheduling, this non-deterministic delay between send command and actual transmission of a radio packet may be in the order of several milliseconds when using a non-preemptive operating system (cf. [82, 111]). For our demands, this delay should be minimized (cf. [123]).

This is one of the reasons why we use the real-time operating system SmartOS [24, 18] (cf. also Section 2.4.4): It operates a 64 bit timeline with a resolution of 1 µs. Noteworthy, SmartOS offers fully preemptive prioritized tasks together with a collaborative resource sharing approach (cf. [17]). Therefore, we defined a sending task with high priority which preempts tasks with lower priority in case a firing packet has to be transmitted. This sending task obviously relies on exclusively shared hardware resources (e.g., SPI bus and RF unit as mentioned in Section 2.4.1 for the SNoW^5 sensor node). However, any of these resources may currently be kept by other tasks (with lower priority), thus the sending task would not be able to duly broadcast its firing packet. The collaborative resource sharing approach of SmartOS comes into play here: The task with lower priority holding at least one resource, which is concurrently requested by the sending task, gets a so-called *dynamic hint* to release this particular resource. By contract, each "hinted" task then releases this specific resource to allow a timely transmission within a well-defined delay. More details about dynamic hinting can be found in [15, 16].

However, this approach will not guarantee the transmission of a firing packet at the intended point in time in any case. Nevertheless, the sending task will be informed about the continued blocking of a resource in advance of the transmission deadline. This information enables the sending task to remedy the situation (e.g., to delay or even to skip the transmission of this current firing packet).

Access Delay Right before the transmission of a packet, the radio unit has to get (exclusive) access to the communication channel to avoid collisions. Depending on the used MAC protocol, this delay is more or less predictable. Due to the underlying schedule, TDMA protocols are more predictable than contention-based CSMA protocols which may implement a (probabilistic) back-off algorithm (cf. Definitions 2.31 and 2.32). Since our self-organizing communication protocol results in a schedule-based TDMA protocol, it does not necessarily rely on Carrier Sense before transmission in general.

Indeed, a node performing Carrier Sense right before its transmission would be able to react on erroneously transmitting nodes accordingly, e.g., by shifting its own time of transmission and thus avoiding collisions. Therefore, using CS right before a transmission could make the extended-Desync protocol more robust against disturbances. However, the RF unit needs a fixed but usually non-negligible amount of time to switch

from the RX mode for Carrier Sense to TX mode for firing (cf. [96]). For instance, the radio chip CC1100 (cf. Section 2.4) typically needs 9.6 µs for this switching operation (cf. [175]). Nevertheless, to minimize the probability of collisions, we enable Carrier Sense for our real-world testbeds: The radio unit is configured to enter TX mode just after a *clear channel assessment* (CCA), i.e., if the RSSI is below a certain threshold and the node is not currently receiving a packet (cf. [175]).

Propagation Delay Even though radio waves in air travel at approximately[3] the speed of light, the propagation of a firing packet takes some time. For instance, a radio wave travels the distance of 300 m in air in about one microsecond.

The communication range of a sensor node in general (cf. Definition 2.6) and of the SNoW^5 sensor node in particular (cf. Section 2.4) is at most a few hundred meters (cf. [26, 68]). Hence, the propagation delay is usually less than 1 µs, which equals the resolution of the SmartOS timeline (cf. Section 2.4.4). Therefore, this delay is impossible to be measured correctly by the deployed hardware, and thus will be neglect.

Transmission Delay Even though the propagation delay is quite small, it takes some time for the sender to transmit the firing packet. However, this delay depends on the packet length as well as on the used data rate. Due to the (common) configuration of the radio unit, both, sender and receiver, are aware of the used data rate.

The length of a firing packet is well-known in advance and integrated into the firing packet (cf. Section 5.6). Thereby, the transmission delay can be estimated accordingly. Since the data rate is limited by the deployed hardware, the packet length has to be reduced to minimize this delay. We will analyze the structure of the information and corresponding data packet to be transmitted in Section 5.5. Subsequently, we will make suggestions on a suitable trade-off between adequate content and transmission delay.

Reception Delay The reception of the firing packet at the receiver's side also takes some time. This delay equals the transmission delay, but it is shifted just by the propagation delay (cf. Figure 5.2). Since packet length and data rate are known by sender and receiver, this delay is also well predictable.

Noteworthy, the reception process is delayed by a certain but fixed time lag due to filtering and circuitry delays of the used radio unit. This phenomenon is further described in [135]. In particular, we measured the delay τ between the sender's SYNC word interrupt and the receiver's SYNC word interrupt for the CC1100 radio unit of our SNoW^5 sensor nodes with an oscilloscope as $\tau_{\text{SNoW}} = 90\,\mu\text{s}$ (cf. [123]). For the comparable CC430 radio unit of the eZ430 Chronos sensor nodes, we measured this delay as about $\tau_{\text{Chronos}} = 20\,\mu\text{s}$. Moreover, this delay τ is independent of the distance between the nodes, but it strongly depends on the configuration of the radio unit, e.g., the radio frequency. Since this constant delay exceeds the propagation delay by far, it is non-negligible and has to be taken into account. For instance, our driver for the radio unit automatically considers this (constant, but hardware dependent) delay τ when receiving a firing packet.

[3] The speed of electromagnetic waves equals the speed of light $c_0 = 299\,792\,\text{km/s}$ in vacuum, but depends on the traversed medium (cf. textbooks on physics like [65]).

Receive Delay The receiving delay is related to the processing of the incoming packet. This includes the transfer to the processing unit (i.e., the microcontroller) as well as the notification of the corresponding task at application layer (cf. Figure 2.6). The receive delay is non-deterministic in general, but mainly depends on similar conditions as the send delay. Indeed, a prompt handling of a firing packet at the receiver is not as time-critical as the adherence to the time of firing by the sender. However, the real-time capabilities of SmartOS (cf. Section 2.4) are beneficial here once again.

Summing up, the real-time operating system SmartOS helps to reduce the send delay as well as the receive delay. The access delay is minimized due to the resulting TDMA protocol, which should provide exclusive access to the medium. Due to the specification of the deployed node hardware, the propagation delay will be neglected. Furthermore, both, transmission delay as well as reception delay, are directly proportional to the packet length, i.e., they are not negligible but predictable. In contrast, the reception delay contains an additional time lag τ introduced by the used radio unit.

So far, we evaluated the communication delays during the exchange of timing information within a firing packet. Additionally, the extended-Desync protocol also relies on accurate timestamps, which will be considered next.

5.4.2 Timestamping

As suggested in Section 5.3.3, each node can achieve timing information about its one-hop neighbors autonomously by timestamping incoming firing packets. According to Eq. (4.11b), the accuracy of the adjustment factor and thus of the next time of firing strongly depends on the accuracy of these timestamps as well as on the accuracy of the propagated timing information. For instance (and provided that Observation 4.10 holds), when node $i \in N$ timestamps w.l.o.g. its predecessor $p(i)$ with an inaccuracy of γ, this would falsify the resulting adjustment factor $\tilde{\varepsilon}_{t_i}$ at least by half of this inaccuracy γ:

$$\begin{aligned}\tilde{\varepsilon}_{t_i} &\overset{4.12b}{=} \frac{t_{s(i)} + \left(t_{p(i)} \pm \gamma\right)}{2} - t_i \\ &= \frac{t_{s(i)} + t_{p(i)}}{2} - t_i \pm \frac{\gamma}{2} \\ &\overset{4.12b}{=} \varepsilon_{t_i} \pm \frac{\gamma}{2}.\end{aligned}$$

One naïve approach is to take all timestamps at application layer (cf. Figure 2.6): An application task would query the system time once the radio protocol notifies a packet reception. Due to the delay caused by the protocol and application stack (cf. Figure 2.6), such timestamps would be quite inaccurate and thus unreliable due to non-deterministic delays of the application software (cf. Section 5.4.1). For instance, using a non-preemptive operating system like TinyOS [82], this delay could be in the order of several milliseconds (cf. [111]). This is not the only reason, why we prefer the preemptive operating system SmartOS [24] for our real-world testbeds as it has integrated timestamping features at IRQ level (cf. Section 2.4). As mentioned before, accurate and precise timing information is mandatory for a proper operation of our self-organizing MAC protocol.

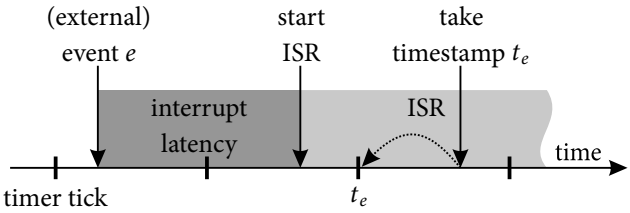


Figure 5.3: Example of interrupt latency during timestamping.

In general, the timestamping of incoming packets may be located at MAC layer instead of application layer (cf. Figure 2.6): Such a low layered timestamping approach may reduce the send delay as well as the receive delay at sender and receiver, respectively. Even though the availability of such timestamping depends on the underlying hardware, its implementation is also advised in [48, 123] – to name but a few. In particular, we use an interrupt-driven timestamping approach in all our testbed implementations: Here, the SNoW^5 sensor node features the radio chip CC1100 (cf. Section 2.4). This radio unit is able to signal every complete reception (or every transmission) of a SYNC word via a dedicated IRQ pin (cf. [175]). Even some microcontrollers with integrated RF transceivers support a similar interrupt concept, although there is no "wired" connection towards the microcontroller. For instance, the CC430 of the eZ430 Chronos (cf. Section 2.4) is such a microcontroller with integrated RF chip, which is able to signal an interrupt on certain (transmission) events.

Since an *Interrupt Service Routine* (ISR) is more privileged than any task at application layer, such an ISR seems ideal to capture timestamps of incoming firing packets. Indeed, even the ISR is executed only after some delay, the so-called *interrupt latency*. This latency is exemplified in Figure 5.3: As already stated in Section 4.1, the local time of each sensor node is discrete. Hence, any external event e will occur in between two subsequent timer ticks. Moreover, the first instruction of the Interrupt Service Routine is started just after some additional delay (*interrupt latency*). Afterwards, within the ISR, the timestamp t_e for the external event e is taken (few ticks) later on. Depending on the actual realization of the microcontroller's interrupt concept (e.g., prioritization of interrupts), this delay may be non-deterministic – especially when an additional *Interrupt Request* (IRQ) with higher priority is pending already.

To consider such a non-deterministic delay in a proper and efficient way, we suggest to utilize the *deep timestamping concept* as specified by Baunach in [18]: The basic idea is to integrate the time management directly into the kernel of the operating system. In particular, if an external event triggers an interrupt then

1. the current timestamp is taken first, and
2. the corresponding IRQ will be executed with access to the previously captured timestamp.

Still, there may be asymmetrical rounding errors during timestamping due to the *discretization of time* (cf. [65]). By configuring the hardware timer adequately, the deep timestamping approach provides an accuracy of $\pm\frac{1}{2}\lambda$ for any time resolution λ (cf. [18]). Besides,

it also supports the autonomous clock drift compensation of a pair of nodes (cf. [19]). Alternatively, the periodical update of neighbor information could be used as well to smooth out non-deterministic delays as described, e.g., in [168].

As mentioned before, the precision of a timestamp strongly depends on the resolution of the underlying local clock. For our real-world testbeds, we use the SmartOS operating system (cf. Section 2.4). This operating system for embedded systems maintains a 64 bit timeline with the resolution of 1 µs[4]. Thereby, the precision is limited to 1 µs, which is sufficient for our purposes. Besides, our simulator is based on a timeline with the same resolution of 1 µs (cf. Section 2.3).

To sum up, an interrupt-driven deep timestamping approach minimizes the time delay between the occurrence of the external trigger event and the timestamp measurement. If the hardware is configured accordingly, the resulting timestamps show an error of $\pm\frac{1}{2}\lambda$ (cf. [18]). This accuracy as well as the precision of our timestamps (of external events) is sufficient for our purposes.

5.5 Neighbor Information

In the last Section 5.4, we analyzed the accuracy and precision of the timestamp measurement. Indeed, due to the hidden terminal problem, a node needs to gain knowledge about the firing time of its two-hop neighbors just by means of the firing packets of its one-hop neighbors. Thus, we have to specify the kind of information as well as the data set which is required for a proper operation of the extended-Desync protocol.

In compliance with Definition 2.10, a firing packet is composed of a header and a payload. While the header provides important control information (like the ID of the sender, a sequence number, or a checksum) of fixed size, the payload of the firing packet contains a set of *neighbor information*, i.e., a certain amount of agreed information about the sender's one-hop neighbors. As already mentioned in Section 4.1, within this section we have to explicitly name the node, whose point of view is relevant for the current analysis. Therefore, we do have to use t_j^i to denote the firing time $t_j^j = t_j$ of node j from the local point of view of node i. In particular, t_j^i denotes node j's reception time of the firing of node i at time t_i^i. This is true while there is a negligible (or at least constant) displacement between sending and receiving time (cf. Section 5.4.1). However, we will use this notation also to express the firing time of two-hop neighbors from a certain node's perspective. Consequently, due to the delays analyzed in Section 5.4.1, we do have to distinguish node i's intended next time of firing ${t^+}_i^i = t_i^+$ from node i's real next time of firing $\hat{t}^+{}_i^i$. Besides, this distinction also holds for node i's intended time of firing t_i^i and node i's real time of firing $\hat{t}_i^i$.

The basis for our discussion of different approaches on how to compile the data required for a proper operation of the extended-Desync protocol, is the sample scenario described in Section 5.5.1. Afterwards, we will start with a naïve approach in Section 5.5.2. Section 5.5.3 and Section 5.5.4 each will present a more sophisticated variant. Finally, Section 5.5.5 concludes this section and reasons one approach to be selected for our further realization and analysis.

[4] The corresponding hardware support is one of the prerequisites to use SmartOS.

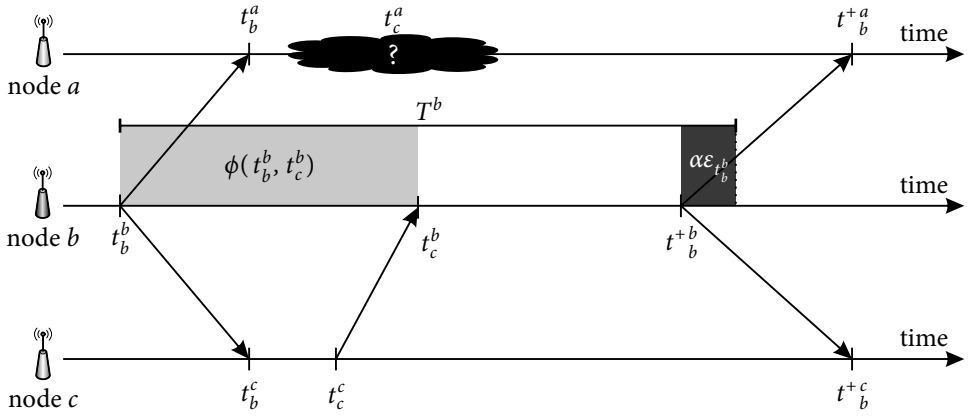


Figure 5.4: Examplary and abstract scenario for the propagation of neighbor information.

5.5.1 Sample Scenario

For the following analysis we rely on Figure 5.4: Like in Section 5.2.2, this linear topology L_3 consists of three nodes, namely node a, node b, and node c. Moreover, node a is not able to directly communicate with node c and vice versa. As introduced in Section 4.1, the term t_b^a denotes the firing time $t_b^b = t_b$ of node b from the local perspective (i.e., registered with the local clock) of node a. Whereas the term T^b denotes the period T from the local point of view of node b (cf. Figure 4.1(b)). Please note that the firings of node a are not illustrated in the figures of this Section 5.5 for the sake of clarity.

In this particular scenario, node b fires at time t_b^b (cf. Figure 5.4). This firing message is received by node a at time t_b^a and by node c at time t_b^c. Next, node c fires at time t_c^c. This firing message is received only by node b at time t_c^b. Since node b needs to adjust its next time of firing, it fires again after period T^b plus the corresponding adjustment factor $\varepsilon_{t_b^b}$ at time t^{+b}_b (cf. Eq. (4.14b)). Please note that the adjustment factor in this example scenario is negative. This firing packet is recognized by node a at time t^{+a}_b and by node c at time t^{+c}_b, respectively. For the following analysis, the main task of node a is – according to its local clock – to estimate the time of firing t_c^a of its two-hop neighbor node c.

5.5.2 Naïve Approach

As a naïve approach, each node just forwards the reception time of the firing packet of all its one-hop neighbors. Indeed, the reception time is always recorded by means of the node's local clock. Following the scenario depicted in Figure 5.4, this means that for instance node b forwards within its firing message at time t^{+b}_b the timestamp t_c^b of its reception of node c's firing message. Consequently, node a will receive this timestamp t_c^b at time t^{+a}_b. Without any additional information about how to interpret this absolute timestamp t_c^b from node b's local clock, node a is unable to transfer the received timestamp from node b's local clock into a timestamp of its own local clock.

In fact, node b could add its initially scheduled next time of firing ${}^{+b}t_b$ to its firing packet, as this point in time is known in due time. However, it is not possible for a node i to always strictly adhere to this intended point in time (cf. Section 5.4.1). Hence, there may be a *firing delay*

$$\delta_{t_i^i} = \hat{t}_i^i - t_i^i \qquad \text{and} \qquad \delta_{{}^{+i}t_i^i} = \hat{t}^{+i}{}_i^i - t^{+i}{}_i^i \tag{5.4}$$

between node i's scheduled (next) time of firing t_i^i (and ${}^{+i}t_i^i$) and its correpsonding real transmission time $\hat{t}_i^i$ (and $\hat{t}^{+i}{}_i^i$, respectively). As $\hat{t}_i^i \geq t_i^i$ (as well as $\hat{t}^{+i}{}_i^i \geq t^{+i}{}_i^i$) is always true, $\delta_{t_i^i} \geq 0$ (as well as $\delta_{{}^{+i}t_i^i} \geq 0$) holds. The receiver should be informed about this latency, which is mainly caused by certain communication delays as described in Section 5.4.1. Especially, the *send delay* is badly predictable but non-negligible. Furthermore, the firing delay even may vary from firing to firing (as indicated by its subscript), i.e., in general $\delta_{t_i^i} \neq \delta_{{}^{+i}t_i^i}$ holds.

By virtue of the timestamping and the interrupt concept introduced in Section 5.4.2, e.g., the SNoW^5 sensor node is able to transmit the real transmission time within firing messages: The CC1100 radio controller signals the transmission of the SYNC word. This interrupt is timestamped by the SmartOS operating system (cf. Section 5.4). This timestamp still can be added (unmodified) to the packet – although the transmission of the packet already is in progress. This procedure performs well despite the very rare case of an TX FIFO underflow (cf. Section 2.4.1), e.g., due to unexpected and arbitrary delays (within the used communication bus system) or due to interrupts with higher priority. In such a disrupted case, the current transmission is aborted. The affected firing packet is corrupted and lost. Thus, each node $i \in N$ always has to add both values to its firing packet, namely its intended time of firing t_i^i and the corresponding real transmission time $\hat{t}_i^i$. This enables each receiver to calculate the firing delay $\delta_{t_i^i}$ of the currently sending node i.

To reduce the required space within a firing packet, just the time difference $\delta_{t_i^i}$ between the timestamp of the SYNC word interrupt and the intended time of firing of the sending node i could be transmitted. However, the computation of this difference complicates the sending process. Besides, even then such packets could be corrupted and lost due to an TX FIFO underflow of the SNoW^5's transceiver. Therefore, we decide to transmit both timestamps as absolute time measurement.

As a result, node a from the sample scenario depicted in Figure 5.5 is able to estimate the firing time t_c^a of node c as

$$t_c^a = {}^{+a}t_b - \delta_{{}^{+b}t_b} - \left({}^{+b}t_b - t_c^b\right) \tag{5.5a}$$

$$\overset{(5.4)}{=} {}^{+a}t_b - \left(\hat{t}^{+b}{}_b - {}^{+b}t_b\right) - \left({}^{+b}t_b - t_c^b\right), \tag{5.5b}$$

where ${}^{+a}t_b$ is measured by node a itself and the timestamps $\hat{t}^{+b}{}_b$, ${}^{+b}t_b$, and t_c^b are taken from the firing packet of node b.

However, Eq. (5.5a) is based on the assumption that ${}^{+a}t_b = \hat{t}^{+b}{}_b$ holds. As already mentioned in Section 5.4 and in [135, 123], this assumption is wrong in general, i.e., ${}^{+a}t_b = \hat{t}^{+b}{}_b + \tau$ technically holds. This delay τ is non-negligible and depends on the used hardware but seems to be fixed (cf. [135]). We did analyze this delay τ empirically for the sensor nodes used in our real-world testbeds: According to our measurements, this delay is $\tau_{\text{SNoW}} = 90\,\mu\text{s}$ for

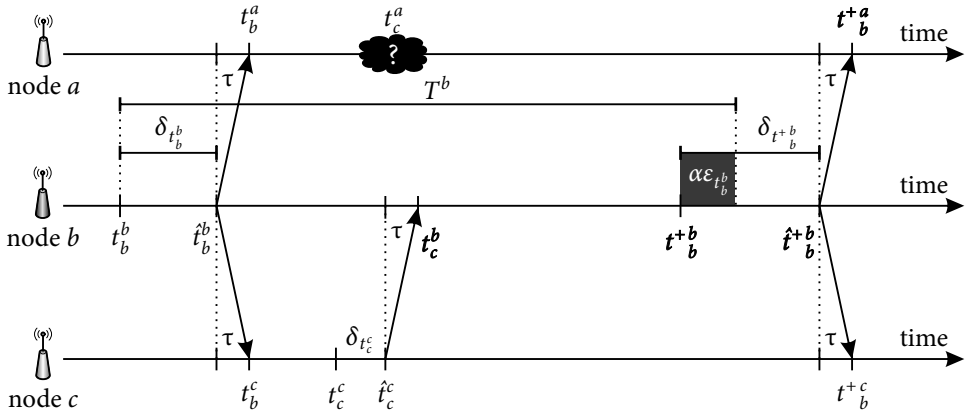


Figure 5.5: The naïve approach regarding the propagation of neighbor information. Bold labels are required by node *a*.

the SNoW^5 sensor nodes and $\tau_{\text{Chronos}} = 20\,\mu\text{s}$ for the eZ430 Chronos (cf. Section 5.4.1). As already mentioned, our driver for the radio unit automatically takes into account this fixed delay when receiving a firing packet. Therefore, we may disregard this delay τ for our further analysis for the sake of clarity.

Indeed, this naïve approach is not very efficient: First, the reception time of the one-hop neighbors and the intended as well as the real transmission timestamp have to be transmitted within each firing packet. According to Eq. (5.5b), these timestamps are $\hat{t}^{+b}_b$, t^{+b}_b, and t^b_c. Moreover, the computation of Eq. (5.5b) is quite complex and cannot be accelerated by the receiver.

As SmartOS operates a 64 bit timeline, each (absolute) timestamp t_x^y is stored as unsigned 64 bit integer of size $l_{t_x^y} = 8\,\text{B}$, i.e., as `uint64_t` within msp430-gcc and as `long` within Java. Consequently, the size required in the firing packet of the sending node b is $l_{t_j^b} = 8\,\text{B}$ for each one-hop neighbor $j \in N_1(b)$ plus $l_{t^{+b}_b} = 8\,\text{B}$ and $l_{\hat{t}^{+b}_b} = 8\,\text{B}$ for the sender's timestamps to enable the receiving node (here: node a) to calculate the time difference $\delta_{t^{+b}_b}$ according to Eq. (5.4). As a result, the size $l_{N_{1\text{naïve}}}$ required in a firing packet of node b for the neighbor information when using the naïve approach in total is

$$l_{N_{1\text{naïve}}} = |N_1(b)| \cdot l_{t_j^b} + l_{t^{+b}_b} + l_{\hat{t}^{+b}_b} \tag{5.6a}$$

$$= |N_1(b)| \cdot 8\,\text{B} + 8\,\text{B} + 8\,\text{B} \tag{5.6b}$$

$$= (|N_1(b)| + 2) \cdot 8\,\text{B}. \tag{5.6c}$$

5.5.3 Phase Shift Approach

According to Eq. (4.13b), the phase shift is technically used for the midpoint approach to compute the node's next time of firing. Therefore, it should be sufficient to just forward the phase shift between the firing node and its one-hop neighbors. Moreover, this data needs not

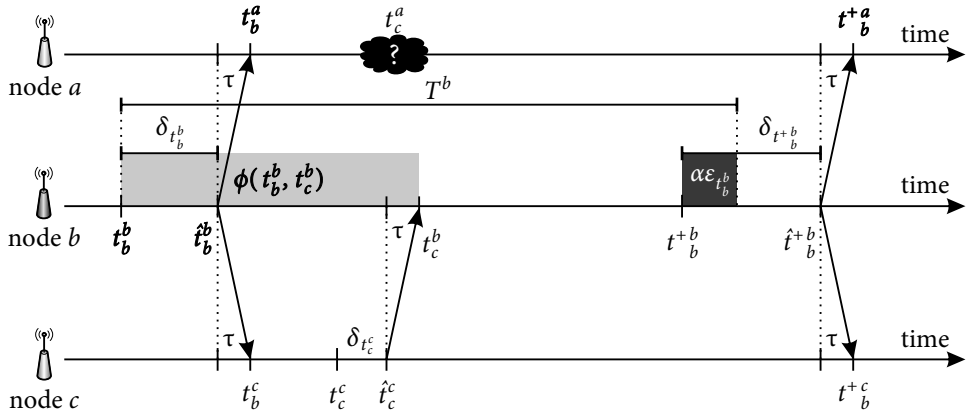


Figure 5.6: The phase shift approach regarding the propagation of neighbor information. Bold labels are required by node *a*.

to be computed separately: Due to the node's computation of its next time of firing, this data already is available at sender's side.

In compliance with Eq. (4.13b), node *b* just transmits the relative phase shift $\phi(t_b^b, t_c^b)$ at its next firing at time t^{+b}_b (cf. Figure 5.6). As long as the receiver is able to remember the last time of firing of the current sender (here: t_b^a), the transmission of the phase shift is sufficient. For instance, node *a* has to recall t_b^a to estimate the firing time t_c^a of its two-hop neighbor node *c* as

$$t_c^a = t_b^a - \delta_{t_b^b} + \phi(t_b^b, t_c^b) \tag{5.7a}$$

$$= t_b^a - \left(\hat{t}_b^b - t_b^b\right) + \phi(t_b^b, t_c^b), \tag{5.7b}$$

where t_b^a was measured by node *a* itself, the phase shift $\phi(t_b^b, t_c^b)$ and the timestamps t_b^b and $\hat{t}_b^b$ are taken from the (previous) firing packet of node *b*.

Since our operating system SmartOS operates an 64 bit timeline with the resolution of 1 µs, a signed 32 bit integer should be sufficient to represent a relative phase shift: We store each phase shift as signed 32 bit integer of size $l_\phi = 4$ B, i.e., as `int32_t` within msp430-gcc and as `int` within Java. Thus, the maximum value of a phase shift, i.e., a delay between two nodes, equates to about 2147.5 s, which is more than half an hour.

This phase shift approach seems to be more efficient than the naïve one from Section 5.5.2: Just the phase shift to each one-hop neighbor has to be transmitted within a firing packet. This data is already available at the sender due to the estimation of its next time of firing. This reduces the computational effort for both, sender and receiver. Due to the fact that we utilize a signed 32 bit integer to store a phase shift, the size $l_{N_1\text{phase}}$ required in a firing packet of node *b* for its neighbor information when using the phase shift approach in total is

$$l_{N_1\text{phase}} = |N_1(b)| \cdot l_\phi \tag{5.8a}$$

$$= |N_1(b)| \cdot 4\,\text{B}. \tag{5.8b}$$

This is more than half the size required by the previous approach (cf. Section 5.5.2). Since the phase shift is always computed according to the current (real) time of firing, additional information about the firing delay is irrelevant here.

However, each node has to store the last time of firing of all its one-hop neighbors to calculate the firing time of its two-hop neighbors: Remember, the information about a node's phase shift always will be propagated within one of the node's upcoming firing packets. For instance, the phase shift $\phi(t_b^b, t_c^b)$ from Eq. (5.7) refers to the firing of node b at time $\hat{t}_b^b$ but will be transmitted by node b at firing time $\hat{t}^{+b}_{\ b}$ at the earliest (cf. Figure 5.6). Therefore, recalling that each (absolute) timestamp t_x^y requires $l_{t_x^y} = 8\,\mathrm{B}$ and the firing delay $\delta_{t_x^y}$ may require $l_{\delta_{t_x^y}} = 4\,\mathrm{B}$, each node has to allocate – at least – memory of size $|N_1(b)| \cdot l_{\delta_{t_x^y}} = |N_1(b)| \cdot 4\,\mathrm{B}$ in addition to the space needed for the firing packet (cf. Eq. (5.8b)). This additional memory requirement equals about the size of the neighbor information in a firing packet.

Furthermore and most disadvantageous, this approach operates with stale information: Any adjustment in the firing time of a one-hop neighbor is deferred. For instance, node b fires at time t_b^b and than adjusts its next time of firing, i.e., $t^{+b}_{\ b} - t_b^b \neq T^b$ holds (cf. Figure 5.6). As a result, node a makes estimations about its next time of firing at time $t^{+a}_{\ b}$ based on the stale information from time t_b^a. We will further analyze this stale information problem in Section 6.2.

5.5.4 Reciprocal Phase Shift Approach

The previous approach from Section 5.5.3 propagates just the phase shift towards one-hop neighbors to save space within the firing packet. However, this approach still

- applies stale information, and
- requires additional memory at each sensor node to store previous times of firings.

Indeed, we want to amend these deficits but to keep the small size of a firing packet at the same time. For this reason, we introduce the *reciprocal phase shift*.

Definition 5.1. Let $\phi_r(t_j^i, t^{+i}_{\ i}) \in \mathbb{N}^+$ be the *reciprocal phase shift* of a node $i \in N$ based on the intended next time of firing $t^{+i}_{\ i}$ and the reception time t_j^i of a firing packet of its one-hop neighbor $j \in N_1(i)$ as

$$\phi_r(t_j^i, t^{+i}_{\ i}) = t^{+i}_{\ i} - t_j^i. \tag{5.9}$$

Please note that the reciprocal phase shift may be greater than the common period T – in contrast to the "normal" phase shift as introduced in Section 4.1.

Observation 5.1. The reciprocal phase shift $\phi_r(t_j^i, t^{+i}_{\ i})$ is always positive, since $t^{+i}_{\ i} > t_j^i$ always holds. It is also of no relevance here, whether node j is one-hop or two-hop neighbor of node i.

Observation 5.2. As already mentioned in Definition 5.1, the reciprocal phase shift could be greater than the common period T. Therefore, the normalization to the common period T as utilized for the phase shift in Eq. (4.2) is not necessary, but instead would be obstructive.

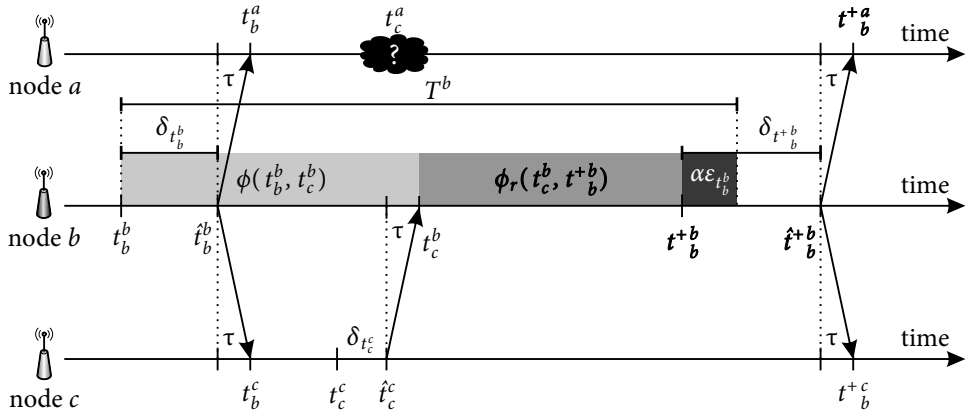


Figure 5.7: The reciprocal phase shift approach regarding the propagation of neighbor information. Bold labels are required by node *a*.

According to Section 4.3, each node is able to set its next time of firing immediately after its current firing[5]. Henceforth, the node's next time of firing is known at the latest after the reception of its successor's firing packet. If a node is receiving a firing packet from a one-hop neighbor, it (additionally) can calculate the reciprocal phase shift towards this one-hop neighbor based on its next time of firing. Due to this relation, the sender has to add its intended next time of firing together with its real transmission time into the firing packet – likewise the naïve approach from Section 5.5.2.

Referring to the sample scenario depicted in Figure 5.7, at time t_c^b node b already knows its next time of firing t^{+b}_b. Therefore, when node b receives the firing packet of node c at time t_c^b, it calculates the reciprocal phase shift

$$\phi_r(t_c^b, t^{+b}_b) = t^{+b}_b - t_c^b. \tag{5.10}$$

Finally, node b transmits the following data within its firing packet at time $\hat{t}^{+b}_b$:

- the $|N_1(b)|$ many reciprocal phase shifts towards its one-hop neighbors,
- its scheduled time of firing t^{+b}_b, and
- its real transmission time $\hat{t}^{+b}_b$.

With it, the receiving node a is able to estimate the firing time t_c^a of node c as

$$t_c^a = t^{+a}_b - \delta_{t^{+b}_b} - \phi_r(t_c^b, t^{+b}_b) \tag{5.11a}$$

$$\overset{(5.4)}{=} t^{+a}_b - \left(\hat{t}^{+b}_b - t^{+b}_b\right) - \phi_r(t_c^b, t^{+b}_b), \tag{5.11b}$$

[5] In Section 6.2.1 we will demonstrate why a node should await the reception of a firing message from its successor before computing its next time of firing.

where ${t^+}_b^a$ is measured by node a itself, the timestamps ${\hat{t}^+}_b^b$ and ${t^+}_b^b$ as well as the reciprocal phase shift $\phi_r(t_c^b, {t^+}_b^b)$ are taken from the firing packet of node b.

Similar to the phase shift approach above, an unsigned 32 bit integer is sufficient for the representation of a reciprocal phase shift. We store a reciprocal phase shift as unsigned 32 bit integer of size $l_{\phi_r} = 4\,\mathrm{B}$, i.e., as `uint32_t` within msp430-gcc and as `int` within Java. Then, the maximum value of a reciprocal phase shift at resolution 1 µs is about 4295 s, which is more than one hour.

In comparison to the phase shift approach from Section 5.5.3, we do need the sender's timestamps ${t^+}_b^b$ and ${\hat{t}^+}_b^b$ to enable the receiving node to calculate the time difference $\delta_{{t^+}_b^b}$ according to Eq. (5.11b). As a result, the size $l_{N_{1\mathrm{reci}}}$ required in a firing packet of node b for the neighbor information when using the reciprocal phase shift approach in total is

$$l_{N_{1\mathrm{reci}}} = |N_1(b)| \cdot l_{\phi_r} + l_{{t^+}_b^b} + l_{{\hat{t}^+}_b^b} \tag{5.12a}$$

$$= |N_1(b)| \cdot 4\,\mathrm{B} + 8\,\mathrm{B} + 8\,\mathrm{B} \tag{5.12b}$$

$$= (|N_1(b)| + 4) \cdot 4\,\mathrm{B}. \tag{5.12c}$$

This is just a little bit more space than required for the neighbor information within a firing packet when implementing the phase shift approach from Section 5.5.3. However, there is no additional memory needed to store timestamps of outdated firing times from one-hop neighbors. In addition, this reciprocal phase shift approach offers the most current timing information within a firing packet.

5.5.5 Summary

The reciprocal phase shift approach from Section 5.5.4 is most beneficial: The naïve approach from Section 5.5.2 just utilizes absolute timestamps for any neighbor information. In contrast, the other two approaches from Sections 5.5.3 and 5.5.4 do reduce the corresponding size of the neighbor information in a firing packet by half. Actually, the phase shift approach even does not require the two absolute timestamps, namely the scheduled time of firing ${t^+}_i^i$ and the real transmission time ${\hat{t}^+}_i^i$ of the sending node i.

However, utilizing the phase shift approach from Section 5.5.3, a receiver does rely on the last firing times of its one-hop neighbors. According to the agreed data types, this approach requires as much memory on the node as required for the neighbor information in a firing packet. Moreover, the reciprocal phase shift from Section 5.5.4 is always based on the node's next time of firing (cf. Definition 5.1). Thus, it bans the majority of sources of stale information. This enables the reciprocal phase shift approach to use more reliable and less stale data than the phase shift approach. If the reciprocal phase shift has to be available for the propagation of neighbor information, we could use the reciprocal phase shift approach also for the estimation of a node's next time of firing as follows: First of all, please note that

$$\phi_r(t_{p(i)}^i, t_i^i) \overset{(4.2)}{=} t_i^i - t_{p(i)}^i \overset{(5.10)}{=} \phi(t_{p(i)}^i, t_i^i) \tag{5.13}$$

for the predecessor $p(i)$ of node $i \in N$ holds. Using the reciprocal phase shift from Section 5.5.4 to calculate the adjustment factor $\varepsilon_{t_i^i}$ of node i with respect to its last time of firing t_i^i at the reception of its successor's firing packet at time $t_{s(i)}^i$ produces

$$\varepsilon_{t_i^i} \overset{4.11}{=} \frac{\phi(t_i^i, t_{s(i)}^i) - \phi_r(t_{p(i)}^i, t_i^i)}{2} \tag{5.14a}$$

$$\overset{(5.13)}{=} \frac{\phi(t_i^i, t_{s(i)}^i) - \phi(t_{p(i)}^i, t_i^i)}{2}. \tag{5.14b}$$

Therefore, we will use the reciprocal phase shift approach from Section 5.5.4 from now on to calculate the node's next time of firing and to be propagated within the corresponding firing packet.

5.6 Information Packing

In the previous section, we optimized the quality of the timing information to be shared with respect to the midpoint approach. Notably, even though it is not necessary to propagate the identifier of a one-hop neighbor for proper operation of the EXTENDED-DESYNC algorithm (cf. Section 4.3), such an identifier simplifies the protocol management and administration. For instance, a receiver has the ability to identify the potential neighbor and to correlate the received timing information with a previously measured one originating from the same sender. Moreover, the node ID as unique attribute (cf. Section 4.1) is quite useful for the verification of the received information as well as for monitoring and debugging purposes (during development). For this reason, we do continue to maintain the identifier as inherent part of the neighbor information. Hence, the space per neighbor information will increase (cf. Section 5.6.1) – irrespective of the realized approaches from the previous section.

Indeed, the size of a firing packet is limited in general: For instance, the CC1100 from our real-world testbed is configured to support packets of a variable packet size with a maximum packet length $l_{\max} = 250$ B in total (cf. [175]). We will use this upper limit as a prerequisite for our analysis within this section. According to Definition 2.10, a (firing) packet contains some control data in the header (cf. Definition 2.10). As a consequence, the remaining payload of a firing packet may be available for neighbor information. In the next sections, we are looking for an approach to use the payload of a firing packet for the transmission of neighbor information in an efficient and space-saving manner.

5.6.1 SmartNET Support

According to our requirements in Section 1.2, we aim for a self-organized and versatile communication protocol for Wireless Networks and Wireless Sensor Networks in particular. To facilitate this objective, we apply the SMARTNET radio stack as described in [18] for our (real-world) implementation – not only for compatibility reasons: The data exchange between existing and potential future SMARTNET based applications (cf. [20, 25, 13, 152, 18],[95, 54, 12][6], and [157][7]) can be realized easily. However, the SMARTNET-compatible radio driver demands

[6] Bachelor theses conducted in conjunction with this work.
[7] Diploma thesis conducted in conjunction with this work.

Symbol	Description	Size	Range
l_{length}	packet length	1 B	0 ... 255
$l_{\text{ID}_{\text{dst}}}$	destination address	2 B	0 ... 65535
$l_{\text{ID}_{\text{src}}}$	source address	2 B	0 ... 65535
l_{flags}	flag field	1 B	0x00 ... 0xFF
l_{app}	application identifier	1 B	0 ... 255
$l_{\hat{t}_i^i}$	real time of firing	8 B	$0 \ldots 2^{64} - 1$
l_{net}	total control data	15 B	

Table 5.2: Inherent parts of the SmartNET-compatible packet header.

$l_{\text{net}} = 15\,\text{B}$ of control data as follows:

- the packet length of size $l_{\text{length}} = 1\,\text{B}$,
- the (next) destination address[8] of size $l_{\text{ID}_{\text{dst}}} = 2\,\text{B}$,
- the (last) source address, i.e., the sender's identifier, of size $l_{\text{ID}_{\text{src}}} = 2\,\text{B}$,
- a flag field, e.g., to signalize a certain message type, of size $l_{\text{flags}} = 1\,\text{B}$,
- the application ID of size $l_{\text{app}} = 1\,\text{B}$, and
- the timestamp $\hat{t}_i^i$ of node i's current transmission[9] of size $l_{\hat{t}_i^i} = 8\,\text{B}$.

Table 5.2 also summarizes the control data of the SmartNET compatible packet header, Figure 5.8 depicts the structure of a firing packet. Hence, not more than

$$l_{\text{data}} = l_{\text{max}} - l_{\text{net}} \tag{5.15a}$$
$$= 250\,\text{B} - 15\,\text{B} \tag{5.15b}$$
$$= 235\,\text{B} \tag{5.15c}$$

per (firing) packet do remain for the payload, namely the neighbor information within this work (cf. also [18]).

Indeed, the SmartNET protocol provides some features which additionally support our extended-Desync protocol: The timestamp of a sender's current transmission can be made available within the control data of the SmartNET compatible packet header automatically. Indeed, this timestamp is required by the reciprocal phase shift approach from Section 5.5.4. Therefore, it is not necessary anymore to add this particular information to the protocol data. Consequently, the protocol overhead issued by the reciprocal phase shift approach can be reduced by $l_{\hat{t}_i^i} = 8\,\text{B}$. As a result, the size $l_{N_{1\text{reci}}}$ required in a firing packet of node $i \in N$ for

[8] Even though we technically just use broadcast messages for our self-organizing MAC protocol.
[9] This process may be automated by setting the corresponding flag in the flag field mentioned above.

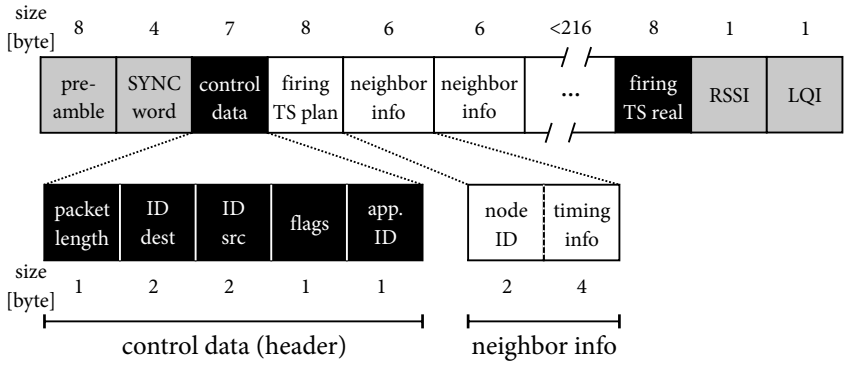


Figure 5.8: The structure of a firing packet. Parts colored in gray are added automatically by the radio unit, parts colored in black are added by the SmartNET-compatible driver.

the neighbor information when using the reciprocal phase shift approach from Section 5.5.4 yields in

$$l_{N_{1\text{reci}}} = |N_1(i)| \cdot \left(l_{\phi_r} + l_{\text{ID}}\right) + l_{t_i^i} \tag{5.16a}$$
$$= \left(|N_1(i)| \cdot 4\,\text{B}\right) + 8\,\text{B} \tag{5.16b}$$
$$= \left(|N_1(i)| + 2\right) \cdot 4\,\text{B}. \tag{5.16c}$$

As indicated before, we do insist on the transmission of node IDs for each corresponding neighbor information. At the SNoW^5 sensor nodes of our real-world testbeds, the node ID of size $l_{\text{ID}} = 2\,\text{B}$ is stored within the volatile memory infomem (cf. Section 2.4.1) as an unsigned short, i.e., as `uint16_t` within msp430-gcc. The SmartOS operating system provides the specific function from Listing 5.1 to enable each node (and each sender, in particular) to easily get its own node ID.

```
1 static inline unsigned short infomem_getNodeID() { return *IM_PNODEID; }
```

Listing 5.1: The SmartOS function to get the node's ID from the SNoW^5's volatile memory infomem.

Hence, the size of each neighbor information increases by $l_{\text{ID}} = 2\,\text{B}$. Consequently, a realization of the reciprocal phase shift approach requires now 6 B per neighbor information, i.e., $l_{\text{ID}} = 2\,\text{B}$ for the identifier of the corresponding one-hop neighbor plus $l_{\phi_r} = 4\,\text{B}$ for the reciprocal phase shift towards this one-hop neighbor (cf. Section 5.5.4). Thus, the size $l_{N_{1\text{reci}}}$, which is required in a firing packet of node i for its neighbor information when using the reciprocal phase shift approach from Section 5.5.4, results in

$$l_{N_{1\text{reci}}} = |N_1(i)| \cdot \left(l_{\phi_r} + l_{\text{ID}}\right) + l_{t_i^i} \tag{5.17a}$$
$$= |N_1(i)| \cdot \left(4\,\text{B} + 2\,\text{B}\right) + 8\,\text{B} \tag{5.17b}$$
$$= |N_1(i)| \cdot 6\,\text{B} + 8\,\text{B}. \tag{5.17c}$$

In comparison to Eq. (5.12), this size now is reduced by $l_{\hat{t}_b^{+b}} = 8\,\mathrm{B}$ (cf. Eq. (5.16)) but is increased by $|N_1(i)| \cdot l_{\mathrm{ID}} = |N_1(i)| \cdot 2\,\mathrm{B}$ for the identifier of node i's neihgbor nodes. The final scheme of such a firing packet when implementing the reciprocal phase shift approach based on the SmartNET-compatible driver is depicted in Figure 5.8.

5.6.2 Naïve Approach

One naïve approach would be that each node always transmits within its firing packet the neighbor information of all its one-hop neighbors once every period. This would be feasible as long as the set of one-op neighbors always is small enough to fit into a single firing packet. As mentioned before, the size of a firing packet in general is limited by certain factors (cf. Section 5.6).

Utilizing the reciprocal phase shift approach, the maximum number $n_{\max}$ of neighbor information per firing packet can be calculated (for the used RF unit CC1100) as

$$n_{\max} = \left\lfloor \frac{l_{\mathrm{data}} - l_{t_i^i}}{l_{\phi_r} + l_{\mathrm{ID}}} \right\rfloor \tag{5.18a}$$

$$= \left\lfloor \frac{235\,\mathrm{B} - 8\,\mathrm{B}}{4\,\mathrm{B} + 2\,\mathrm{B}} \right\rfloor \tag{5.18b}$$

$$= \left\lfloor \frac{227\,\mathrm{B}}{6\,\mathrm{B}} \right\rfloor \tag{5.18c}$$

$$= \lfloor 37.83 \rfloor \tag{5.18d}$$

$$= 37. \tag{5.18e}$$

This limitation is valid for all sensor nodes of our real-world testbeds (cf. Section 2.4). Nevertheless, this maximum number seems to be a sufficiently large number of potential one-hop neighbors. However, there may be multi-hop topologies with nodes having more than $n_{\max} = 37$ one-hop neighbors. Moreover, the transmission delay as well as the reception delay (cf. Section 5.4.1) are directly proportional to the length of the firing packet. This means that a smaller number of neighbor information effects shorter delays (cf. Table 5.1). Therefore, it may be sensible to limit the quantity of neighbor information which has to be propagated within a single firing packet. In this regard, the appropriate subset of potentially available neighbor information (i.e., information from a subset of the available one-hop neighbors) has to be selected. Next, we will discuss in the following sections some selection strategies.

5.6.3 Fixed Size Subset Approach

One option could be to define a fixed upper bound $n_{\mathrm{fixed}} \in \mathbb{N}^+$ for the quantity of neighbor information to be transmitted within a single firing packet. Certainly, this fixed upper bound is naturally restricted by the maximum packet length $l_{\max}$. In accordance with Eq. (5.18), for our real-world testbed holds

$$n_{\mathrm{fixed}} \quad \leq \quad n_{\max} \quad \overset{5.18}{=} \quad 37. \tag{5.19}$$

If the corresponding neighbor information is selected progressively without any duplicates, node i has transmitted the neighbor information of all its one-hop neighbors $N_1(i)$ after at least

$$m_{\text{fixed}} = \left\lceil \frac{|N_1(i)|}{n_{\text{fixed}}} \right\rceil \tag{5.20}$$

firing packets.

For the following considerations, we assume that the system is unlikely to change, in particular, there will be no topology dynamics. For instance, let $|N_1(i)| = 100$, i.e., node $i \in N$ has 100 one-hop neighbors, and let $n_{\text{fixed}} = 8$, i.e., the maximum capacity of neighbor information for each firing packet was restricted to 8. Then node i has propagated the neighbor information of all its one-hop neighbors $N_1(i)$ after at least

$$\begin{aligned} m_{\text{fixed}} &\overset{(5.20)}{=} \left\lceil \frac{100}{8} \right\rceil \\ &= \lceil 12.5 \rceil \\ &= 13 \end{aligned}$$

firing packets.

This latency is inversely proportional to the maximum quantity of neighbor information per firing packet (cf. Eq. (5.20)). Thus, a smaller quantity n_{fixed} of neighbor information per firing packet results in a higher number of firing packets required to gain total knowledge about another node's one-hop neighbors. Certainly, this proposition is valid only if the one-hop neighborhood of the (sending) node is unlikely to change.

5.6.4 Fixed Percentage Approach

Likewise the fixed upper bound for the quantity of neighbor information, a certain percentage $n_{\text{percent}} \in [0,1]$ of the available neighbor information could be transmitted. Since $n_{\text{percent}} = 0$ means that no neighbor information will be transmitted at all, this endpoint of the interval is not considered any further.

Nevertheless, and likewise the fixed upper bound n_{fixed}, the maximum percentage n_{percent} is also naturally restricted for a node $i \in N$. Additionally, the maximum percentage n_{percent} depends on the individual size of the node's one-hop neighborhood $|N_1(i)| \geq 1$:

$$n_{\text{percent}} \leq \min\left\{ \frac{n_{\max}}{|N_1(i)|}, 1 \right\}. \tag{5.21}$$

If for instance $|N_1(i)| = 100$, i.e., node i has 100 one-hop neighbors, then for our real-world testbed (cf. Section 5.6) holds

$$\begin{aligned} n_{\text{percent}} &\overset{(5.21)}{\leq} \min\left\{ \frac{37}{100}, 1 \right\} \\ &\leq \min\{0.37, 1\} \\ &\leq 0.37. \end{aligned}$$

This means that node i is able to transmit a maximum of 37 % of its 100 one-hop neighbors per single firing packet. Once more, if the corresponding neighbor information is selected progressively without duplicates, node i has transmitted the neighbor information of all one-hop neighbors $N_1(i)$ after at least

$$m_{\text{percent}} = \left\lceil \frac{|N_1(i)|}{\lfloor |N_1(i)| \cdot n_{\text{percent}} \rfloor} \right\rceil \tag{5.22}$$

firing packets.

For instance, let $|N_1(i)| = 100$, i.e., node i has got 100 one-hop neighbors, and let $n_{\text{percent}} = 0.125$, i.e., the maximum percentage of the node's one-hop neighborhood for each firing packet equals 12.5 %. Then node i has propagated the neighbor information of all its one-hop neighbors after at least

$$\begin{aligned} m_{\text{percent}} &\overset{(5.22)}{=} \left\lceil \frac{100}{\lfloor 100 \cdot 0.125 \rfloor} \right\rceil \\ &= \left\lceil \frac{100}{12} \right\rceil \\ &= 9 \end{aligned}$$

firing packets.

Again, the packet count is inversely proportional to the maximum number of neighbor information per firing packet (cf. Eq. (5.22)). Thus, a smaller maximum percentage of the node's one-hop neighborhood as neighbor information per firing packet n_{percent} results in a higher number of firing packets needed to gain sufficient knowledge about another node's one-hop neighbors. This approach facilitates a dynamic trad-off between the percentage of neighbor information and the size of a single firing packet. Certainly, this proposition is valid only if the one-hop neighborhood of the sending node is unlikely to change.

However, in contrast to the fixed size subset approach with the fixed upper bound n_{fixed} (cf. Section 5.6.3), the maximum percentage n_{percent} strongly depends on the individual size of the node's one-hop neighborhood (cf. Eq. (5.21)). Consequently, any modification of the node's one-hop neighborhood as basic part may disable the above statement. As a result, the fixed percentage approach is not constant but necessitates appropriate subset updates. For instance, let $|N_1(i)| = 100$, i.e., node $i \in N$ has got 100 one-hop neighbors, and let $n_{\text{percent}} = 0.1$, i.e., the maximum percentage of the node's one-hop neighborhood for each firing packet equals 10 %. As a result, node i will transmit the neighbor information of $100 \cdot 0.1 = 10$ one-hop neighbors per firing packet. If the size of node i's one-hop neighbors reduces to 50, the number of neighbor information of just $50 \cdot 0.1 = 5$ one-hop neighbors per firing packet will be transmitted. This different number of neighbor information results in a different size of the firing packet. Consequently, transmission and reception delays also differ. Therefore, we do not consider this approach to be applicable enough for our purpose.

5.6.5 Summary

The naïve approach from Section 5.6.2 is not sufficient for our purposes at all. Moreover, for our (real-world testbeds implementation of the) EXTENDED-DESYNC protocol, we do prefer the fixed size subset approach with the fixed upper bound n_{fixed} from Section 5.6.3 to the

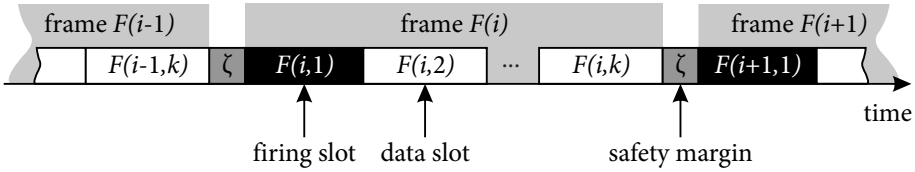


Figure 5.9: The structure of a frame of our EXTENDED-DESYNC protocol. The frames are slotted and divided by a safety margin ζ.

fixed percentage approach with the fixed percentage n_{percent} from Section 5.6.4: Indeed, the fixed size subset approach may not use the available resources, mainly the node's memory, in an optimal way. Especially, when the number of one-hop neighbors is not divisible without remainder by the selected, fixed size of the subset.

However, the fixed size subset approach is more predictable and thus more reliable than the fixed percentage approach: The fixed percentage approach strongly depends on the (varying) size of the node's one-hop neighborhood. Therefore, the quantity of neighbor information to be transmitted within the node's current firing packet has to be recalculated at any dynamic change in the node's one-hop neighborhood. Thus, not only the size of a firing packet but also its transmission delay as well as reception delay are quite unstable. Utilizing the fixed size subset approach, especially these time consuming calculations per period can be omitted. This procedure results in a more stable and predictable behavior notwithstanding topology dynamics.

5.7 Frame Structure

In the last section, we discussed several approaches regarding the packet structure of a firing message with a special focus on the neighbor information. However, it is in general essential that sensor nodes are able to exchange additional (sensor) data to accomplish a common task – depending on the particular WSN application. Thus, the communication protocol has to accommodate application data as well.

As already proved in Lemma 4.13, a collision-free communication period T has to support at least $\max_{i \in N}\{|N_C(i)|\}$ firing packets for a network N. Consequently, the period T is the limiting factor of the midpoint approach. For this purpose, each node $i \in N$ allocates one single *frame* $F(i)$ of size $l_{F(i)} = |F(i)| < \phi(t_i, t_{s(i)})$ per period. Similar to other MAC protocols implementing a TDMA scheme (e.g., LMAC [187] or Crankshaft [77]), each frame $F(i)$ may be subdivided into k *slots* $F(i,j)$ with $j \in \{1,\ldots,k\}$. Figure 5.9 illustrates the structure of such a frame.

The first slot $F(i,1)$ of the frame $F(i)$ of each node i is assigned to its firing message (cf. Definition 3.2 as well as Section 5.6). Thus, we also call this slot *firing slot*. The remainder of the frame is left to the free use of the application(s), i.e., the remaining slot(s) $F(i,k)$ for $k \geq 2$ of node i can be used for application data transmission. This means that the firing slot marks the start of the node's transmission frame as depicted in Figure 5.9. Utilizing the fixed size subset approach for the information packing from Section 5.6.3 as suggested in Section 5.6.5,

the size of the firing slots is equal for each node. Notably, for $k = 1$ the corresponding frame will contain just one single slot, namely the firing slot. Consequently, the length of a frame equals the length of the firing slot, i.e., $l_{F(i)} = |F(i,1)|$. However, it is not possible to transmit any application data then.

Furthermore, we introduce a *safety margin* ζ to compensate for potential collisions due to (re)joining nodes, clock drifts (cf. Section 7.1.1), or other indeterminate delays based on software defect or hardware failure (cf. also [96]). This safety margin precedes the first slot of a node's frame as shown in Figure 5.9. Indeed, the size of this safety margin $|\zeta|$ is a trade-off between appropriateness and network utilization: If the safety margin is smaller than reasonable, potential collisions may not be compensated, and if the safety margin is bigger than necessary, the network is utilized not in an efficient way. Therefore, for the size of this safety margin we propose at least half of the size of the firing slot $|F(i,1)|$ of node $i \in N$.

Further information about this frame structure of our EXTENDED-DESYNC protocol can be found in [121]: This work also outlines a short and theoretical analysis of energy-related and temporal aspects regarding the EXTENDED-DESYNC protocol. As a conclusion, the author in [121] argues not to relate the size of the (firing) slot of the frame to the variable size of the node's one-hop neighborhood. This is another reason against the implementation of the fixed percentage approach from Section 5.6.4

5.8 Practical Issues

Mainly due to the experiences with our real-world testbeds, we discovered some curious system behavior – although the nodes still did fulfill the qualifications of the EXTENDED-DESYNC protocol, like unique identifiers, periodical firings, and a period of sufficient length. This behavior does not fit well to our objective of robust and adaptable MAC protocol (cf. Section 1.2). Nevertheless, this behavior is reproducible and thus could be analyzed in detail.

In summary, we identified a strong correlation with topology dynamics. Therefore, we will have a closer look at nodes, which are re-joining the network, and at nodes, which are (seemingly) leaving the network. Based on the results from [68, 167][10], and adapted from our work in [128], we introduce additional protocol parameters to make our EXTENDED-DESYNC protocol

- more robust against topology dynamics and
- more adaptable with respect to nearby nodes starting up concurrently.

In particular, we will provide a listening time in combination with a node-specific waiting time in Section 5.8.1 as well as an expiration time in Section 5.8.2.

5.8.1 Nodes In

For some scenarios, our system initially did behave quite badly, i.e., the system hardly desynchronized. This behavior could be observed mainly for real-world testbeds, when some nodes where switched on concurrently, e.g., using a switchable multi-outlet power strip. Due to

[10] Diploma theses conducted in conjunction with this work.

such a concurrent start up, all these nodes simultaneously try to follow the implemented EXTENDED-DESYNC protocol. In particular, without any variation, each node immediately tries to broadcast its first firing packet. As a consequence, the firing packets are most likely transmitted at the very same time. Depending on the nodes' communication ranges and thereby the underlying topology, in worst case all firing packets could collide. Especially in a single-hop topology, not a single node would gain knowledge about the existence of any other neighbor. If there is no variation or extraordinary "occurrence", like clock drift or topology dynamics, there will be no change in behavior, and thus the firing packets will collide permanently.

In the following, we will exemplify and analyze this issue in detail by means of the simple complete graph C_2 consisting of two nodes, node a and node b, where node a is within the communication range of node b – and vice versa. In addition, both nodes are starting up concurrently, and thus both will broadcast their (first) firing packet at the same time. As already mentioned, in doing so, the firing packets will collide. Since both nodes here do not receive further information, e.g., about other neighbors, each node still assumes to be alone. Again, each node waits one period, during which no other neighbor node will be detected. Thus, node a and node b will broadcast their firing packets again at the same time, and, once more, both firing packets will collide. Since none of both nodes did receive further information, repeatedly, both firing packets will be transmitted at the same time and will collide again. Consequently, both nodes never will detect each other. Instead, each node will assume to be the only node within this particular network – at least as long as there is no "randomness", like clock drifts or topology dynamics.

Therefore, we introduce some randomness in terms of a random amount of time $\Delta t_{\text{rand}} \in [0, T)$ following a discrete uniform distribution. Each node now has to additionally wait this random amount of time Δt_{rand} before transmitting its first firing packet. In this regard, we had to install at each node a Pseudo Random Number Generator (cf. Section 6.4.1), which uses the unique node ID as seed to generate such random numbers. The probability of nodes still firing at the same time decreases with an increasing amount of periods. Indeed, the introduction of randomness solves the issue on concurrently starting nodes.

However, this randomness was not sufficient for any scenario: We could observe some "restlessness" of the system during settling time, especially when nearby nodes (i.e., potential one-hop neighbors) were joining the network node by node within too closely timed intervals. This means that appropriate parts of the network were not stabilized enough to tolerate these topology dynamics of quick succession. To temper such "restlessness", we introduced an additional *listening time*

$$\Delta t_{\text{listen}} = n \cdot T \tag{5.23}$$

with $n \in \mathbb{N}^0$ at the start up of each node. In particular, each node has to listen first for a certain amount of periods, the so-called *listening periods*, before it is allowed to broadcast its first firing packet.

On the one hand, this listening time at start up enables a joining node to gather reliable information about its neighborhood and its constraint graph (cf. Definition 2.28), respectively. Consequently, the node is enabled to join "smoother", i.e., to select an appropriate point in time for its first firing beforehand. On the other hand, such a passiveness (cf. Sec-

tion 2.4.5) prevents the node for the specified listening time from transmitting messages and from exchanging information, respectively.

Especially in urgent cases, when for instance control values for actuators have to be transmitted at the right time, any delay is undesired or even counterproductive. Hence, a trade-off is desired between the point in time, when a node is allowed to transmit its first firing packet since its start up, and a faster but more controlled desynchronization process. For our simulations and testbeds, we identified a number of three listening periods, i.e., $\Delta t_{\text{listen}} = 3 \cdot T$, as a proper and useful initial value for a node's start up phase.

In summary, the start up phase of a node implementing our back-off strategy from above is as follows:

1. The node is powered on.
2. The node sets its individual waiting time $\Delta t_{\text{rand}} \in [0, T)$ according to its node ID (cf. Listing 5.1).
3. The node next listens for a total of $\Delta t_{\text{listen}} + \Delta t_{\text{rand}} = 3 \cdot T + \Delta t_{\text{rand}}$ before the transmission of its first firing packet.
   a) If the node receives at least one firing packet from another node during this time span, it matches the received neighbor information and chooses the phase of its first time of firing as shown in Listing 5.3 on Page 107. The further progress follows the EXTENDED-DESYNC algorithm as described in Section 5.3.
   b) Else, when this listening time elapsed without receiving any single firing packet, the node immediately broadcasts its first firing packet and now listens for $T + \Delta t'_{\text{rand}}$, i.e., one period T plus another random waiting time $\Delta t'_{\text{rand}} \in [0, T)$.

   The further progress follows 3a or 3b, depending on whether firing packets of other nodes could be received during the individual listening time – or not.

5.8.2 Nodes Out

From some of our real-world testbeds, we obtained further strange outcomes. In particular, we observed some outliers of unsteady and unsettled behavior caused by unreliable links, i.e., usually bidirectional links between two nodes which just sporadically behave like unidirectional links. By reason of such an unreliable link, a node (with just local view) may not receive anymore firing packets (accompanied by neighbor information (cf. Section 5.5)) of formerly one-hop neighbors. Usually, if the (expected) firing packet of an already known neighbor is not received within the next period, this still-neighbor will be removed immediately from the corresponding list. As a consequence, this neighbor will not be considered as relevant node for the adjustment function of the midpoint approach – at least till a firing packet of this excluded neighbor is received once again. Due to the midpoint approach, the receiving node will adjust its next time of firing accordingly if the excluded neighbor was a phase neighbor.

Due to the unreliable link, it is very likely that the seemingly exited neighbor is re-joining the network within one of the subsequent periods. If this re-joining neighbor becomes phase neighbor again, the receiving node once more will have to adjust its next time of firing accordingly. This issue is even multiplied when the receiving node is also phase neighbor of

another node or even a multiple phase neighbor (cf. Definition 4.4). As a result, nodes of (at least parts of) the network will have to adjust their next time of firing as well. As a consequence, the system shows a "restless" behavior. Depending on how often and for how long such an unreliable link persists, the corresponding part(s) of the network may be not stable (with regards to desynchrony) for a certain amount of time. Especially, such topology dynamics are not easy to cope with at frequent intervals.

To introduce more stability, each node $i \in N$ adds to each of its (one-hop as well as two-hop) neighbors $j \in N_1(i) \cup N_2(i)$ an *expiration time*

$$\Delta t_{\text{expire}}(j) = n \cdot T \tag{5.24}$$

with $n \in \mathbb{N}^0$. This means that if an already known neighbor $j \in N_1(i) \cup N_2(i)$ is not received within this amount of consecutive periods, its firing time will be extrapolated by one period each. Hence, the neighbor's time of firing will be kept virtually up-to-date. In particular, the corresponding reciprocal phase shift will be kept. In fact, a neighbor will be removed from a node's neighbor list after the specified number of consecutive periods without any reception. Nevertheless, the abrupt absence of a neighbor (and a phase neighbor in particular) does not necessarily provoke prompt adjustments of firing times when utilizing the expiration time. For a better adaptability according to the underlying topology, we introduced the expiration time for one-hop neighbors as well as for two-hop neighbors, each to be set independently but node-specific.

If the seemingly exited and virtually kept neighbor will re-join the network during the expiration time, there may be a little deviation between the "real" time of firing and the "virtual" time of firing – especially in a not yet (perfect) desynchronized network. Moreover, the impact on the system of such small differences in the time of firing is far less than the occurrence of unreliable links without expiration time. For our simulations and testbeds, we identified an expiration time of about three periods, i.e., $\Delta t_{\text{expire}}(j) = 3 \cdot T$, as a proper and useful initial value for one-hop as well as two-hop neighbors $j \in N_1(i) \cup N_2(i)$ of node $i \in N$. This setting smooths out the "restless" behavior based on unreliable links quite well. Likewise the listening periods from Section 5.8.1, the selected value for the expiration time provides a good trade-off between latency and fault-tolerance.

5.9 Summary

In this chapter, we described our self-organizing MAC protocol EXTENDED-DESYNC. In particular, we improved the midpoint approach from Section 4.3 to make it applicable for multi-hop topologies. In this regard, the main challenge was to solve the hidden terminal problem (cf. Section 5.2). Due to the fact that already available approaches, which do solve the hidden terminal problem, are not sufficient for our purposes, we developed the phase shift propagation approach in Section 5.3: The objective of the phase shift propagation is to enable each node to create its constraint graph solely based on local information (cf. Section 5.3.2) . With it, each node gains knowledge about its two-hop neighborhood required for a self-organizing and collision-free communication. Enriching the neighbor information with additional information about the reception time of the neighbor's firing messages enables each node to consider its two-hop neighbors for the calculation of its next time of firing accordingly, and thus to solve the hidden terminal problem.

The quality as well as the steadiness of our protocol strongly depends on accurate and precise timing information. In Section 5.4 we analyzed potential communication delays as well as timestamping mechanisms. Particularly, the exchange of accurate timestamps is the key factor of our EXTENDED-DESYNC protocol in this regards. Based on this, we described the neighbor information to be exchanged in Section 5.5: As a result, each node broadcasts its reciprocal phase shift (cf. Definition 5.1) to allow the receiving nodes an accurate and precise estimation of transmission times of their corresponding two-hop neighbors.

As envisioned in Section 5.6, the information about a node's whole one-hop neighborhood may be too large to fit into one single firing packet. As a consequence, the information about a node's one-hop neighborhood has to be distributed among several firing packets. After the introduction of several approaches about which information shall be considered within a firing packet and how, we concluded to implement the fixed size subset approach from Section 5.6.3. Indeed, it may take several periods for a receiving node to collect all the information required when implementing this approach. However, the fixed size of the firing packet allows for a more predictable communication behavior. This benefit is crucial, especially, when the period is divided into frames to also accommodate application data as described in Section 5.7. Based on our practical experiences, we further enhanced our EXTENDED-DESYNC protocol in Section 5.8. In particular, we introduced a waiting time for (re)joining nodes and an expiration time for neighbors which (seemingly) left the network.

Next, we present the algorithm to calculate the next time of firing of a node $i \in N$ of our EXTENDED-DESYNC protocol. In particular, we inspect the event on sending a firing packet by node i, and the event on receiving a firing packet from a neighbor node by node i. The firing case is presented in Listing 5.2. Here, we assume that node $i \in N$ has to transmit its firing packet: This means that the previously set firing timer expires, i.e., the next time of firing is reached (cf. Line 2). Consequently, the "old" next time of firing $t^{+}{}_{i}^{i}$ becomes the current time of firing t_i^i in Line 4. As a precaution[11], after transmitting its firing packet in Line 5 at time $\hat{t}_i^i$, node i schedules its next time of firing at time $t^{+}{}_{i}^{i}$ in Line 9. In Line 6, the node's real transmission time $\hat{t}_i^i$ is extracted from the just transmitted firing packet (cf. Section 5.6.1) to calculate the firing delay $\delta_{t_i^i}$ for monitoring purposes (cf. Line 7). Node i will use the precautionary scheduled next time of firing in case it will not receive any other firing packet until then.

```
1  // upon firing of node i
2  if (firingTimerExpired()) {
3    setFirstPacketReceived(true);
4    t_i^i = t^+_i^i; // "old" next firing time now is current firing time
5    transmit(firingPacket); // blocking send command
6    t̂_i^i = getRealTransmissionTime(firingPacket);
7    δ_{t_i^i} = (t̂_i^i − t_i^i);
8    t^+_i^i = t_i^i + T;
9    setFiringTimer(t^+_i^i); // precautionary scheduling
10 }
```

Listing 5.2: The calculation of the next time of firing $t^{+}{}_{i}^{i}$ of node $i \in N$ as implemented in our EXTENDED-DESYNC protocol when node i is sending its firing packet.

[11] The exact reasons for this precaution will be explained in detail in Section 6.2.1

The receiving case is presented in Listing 5.3. Here, node $i \in N$ is receiving at least one firing packet from a neighbor node before the expiration of its firing timer: The received firing packet may be the first one after node i's own transmission – or not. Assuming, the currently received firing packet is the first firing packet after the transmission of its own firing packet (cf. Line 2) at time $t^i_{s(i)}$ (cf. Line 4). According to Definition 4.2, this neighbor node is the successor $s(i)$ of node i. As stated in Section 5.5.5, node i then has to update the precautionary scheduled next time of firing $t^{+}{}^{i}_{i}$ in Lines 5 to 8 according to Eqs. (4.14) and (5.14a). If node i receives additional firing packets from further neighbor nodes (cf. Lines 9 to 12), node i records the particular reception time (cf. Line 10) of this potential predecessor (cf. Definition 4.2). Moreover, it also calculates the corresponding reciprocal phase shift $\phi_r(t^i_{p(i)}, t^{+}{}^{i}_{i})$ in Line 11 based on the updated next time of firing $t^{+}{}^{i}_{i}$ as described in Section 5.5.4.

```
1  // upon receiving a firing packet
2  if (isFirstPacketReceived()) {
3    setFirstPacketReceived(false);
4    t^i_{s(i)} = now();
5    φ(t^i_i, t^i_{s(i)}) = (t^i_{s(i)} − t^i_i);
6    ε_{t^i_i} = ( φ(t^i_i, t^i_{s(i)}) − φ_r(t^i_{p(i)}, t^i_i) ) /2;
7    t^{+}{}^{i}_{i} = t^i_i + T + α · ε_{t^i_i}; // overrides the next time of firing
8    setFiringTimer(t^{+}{}^{i}_{i});
9  } else {
10   t^i_{p(i)} = now();
11   φ_r(t^i_{p(i)}, t^{+}{}^{i}_{i}) = (t^{+}{}^{i}_{i} − t^i_{p(i)});
12 }
```

Listing 5.3: The calculation of the next time of firing $t^{+}{}^{i}_{i}$ of node $i \in N$ as implemented in our EXTENDED-DESYNC protocol when node i is receiving firing packets.

Nevertheless, we could observe another phenomenon, which is inherent to the primitive of desynchronization, namely the *stale information problem*: The received data about a neighbor sometimes is "stale", i.e., the information obtained from a received firing packet may be obsolete at the time of its utilization. Consequently, this information may be unreliable or even invalid. Such stale information has a strong impact on the convergence behavior of the system (cf. [126]). However, in this regard our EXTENDED-DESYNC protocol is limited in terms of robustness as well as stability due to the stale information problem (cf. Section 6.2). Since this problem is inherent to the primitive of desynchronization and already appears in single-hop topologies but is intensified in multi-hop topologies, further enhancements are necessary. Therefore, we developed EXTENDED-DESYNC^{+} which is explained in detail in the next chapter.

Chapter 6

The extended-Desync$^+$ Protocol

Abstract

This chapter describes the EXTENDED-DESYNC$^+$ protocol, which improves the self-organizing MAC protocol EXTENDED-DESYNC from Chapter 5. As already indicated in Section 5.9, the applicability of the EXTENDED-DESYNC protocol as realization of the midpoint approach for multi-hop topologies is restricted by the *stale information problem*. We will repeat the formulation of this problem in Section 6.1. After this short motivation, we will analyze the stale information problem for single-hop as well as multi-hop topologies in Section 6.2. In Section 6.3, we will present the *refractory threshold* as additional protocol parameter to solve this problem especially for multi-hop topologies. A short excursion on Pseudo Random Number Generator in Section 6.4 explains the particular implementation for our real-world testbeds as well as for our simulator EXTDESIMC. Finally, Section 6.5 summarizes the features of the resulting EXTENDED-DESYNC$^+$ protocol.

6.1 Motivation

In Chapter 5 we introduced the phase shift propagation to extend the midpoint approach for use as self-organizing MAC protocol for multi-hop topologies. The resulting EXTENDED-DESYNC protocol seems to meet all demands from Section 1.2. Initially, our EXTENDED-DESYNC protocol is not limited to certain topology types – in contrast to some other MAC protocols based on the primitive of desynchronization (cf. Section 5.1). Finally, numerous simulations as well as experiments of our EXTENDED-DESYNC protocol for various single-hop and multi-hop topologies, including joining or leaving nodes, always reached the stable state of desynchrony (cf. [122, 123, 128] and [68, 167][1]).

Surprisingly, for certain multi-hop scenarios, we observed a remarkable behavior of the overall network, and, in compliance with Definition 2.1, of the nodes: The system did not reach the stable state of desynchrony. Quite the contrary, some nodes alternated between two specific values (according to their relative phase). Moreover, it seems that the system presumably will never reach the stable state of desynchrony. Interestingly, we could neither observe nor provoke such a behavior for any single-hop scenario.

A deeper analysis (cf. Section 6.2.2) of such a multi-hop topology with our simulator EXTDESIMC (cf. Section 2.3.2) enables us to identify the reason for this alternating behavior, namely the so-called *stale information problem*. This problem is inherent to the primitive of desynchronization and yet known for single-hop topologies. However, this problem – to the best of our knowledge – has not been investigated for multi-hop topologies, yet.

[1] Diploma theses conducted in conjunction with this work.

Consequently, the problem's impact and its effect on the system behavior is also still under-researched. Before we present our approach to solve the stale information problem in Section 6.3, we first describe this problem for single-hop as well as for multi-hop topologies in the next section.

6.2 Stale Information Problem

One characteristic of our self-organizing EXTENDED-DESYNC protocol is the ability of each node to work with just locally available information. In fact, some information to solve the hidden terminal problem is "self-provided" (e.g., information about a one-hop neighbor), whereas some is "self-acquired" (e.g., information about a two-hop neighbor). In this regard, the EXTENDED-DESYNC protocol implements the phase shift propagation approach where each node propagates information about its one-hop neighbors within its firing packets (cf. Section 5.3). Consequently, neighbor information of a two-hop neighbor (cf. Section 5.5) which has been propagated by a one-hop neighbor usually becomes "stale". This means that the neighbor information, which a node is obtaining from received firing packets, may be obsolete, and thus unreliable or even invalid.

Even more remarkable, the problem of "stale" information already exists in single-hop topologies (see Section 6.2.1): Here, this problem just impacts the convergence rate of the system, i.e., the stable state of desynchrony still will be reached eventually – however later than necessary. As mentioned before, the effect of this problem is intensified for multi-hop topologies (see Section 6.2.2): Here, "stale" information additionally impacts the convergence behavior (of particular nodes), i.e., the stable state of desynchrony may not be reached at all (cf. Section 5.9).

6.2.1 Single-Hop Topology

The main characteristic of the midpoint approach as underlying concept of our EXTENDED-DESYNC protocol is each node's ability to autonomously calculate its next time of firing. In fact, when a node $i \in N$ utilizes Eqs. (4.14) and (5.14) to calculate its next time of firing t_i^+, this computation is based on "stale" information: Each node autonomously operates on just locally available data. Hence, any adjustment of a node's next time of firing is neither subject to approval by other nodes nor has the node's adjustment to be announced or registered in advance: Assuming, node $i \in N$ transmits its firing packet at time t_i and immediately calculates its next time of firing t_i^+ according to Eq. (4.14). Meanwhile, each phase neighbor $p(i)$ and $s(i)$ may already have autonomously adjusted its individual next time of firing $t_{p(i)}^+$ and $t_{s(i)}^+$, respectively. Therefore, the phase shifts $\phi(t_{p(i)}, t_i)$ and $\phi(t_i, t_{s(i)})$ may be based on stale information. Hence, the estimation of the next time of firing of node i utilizes old, and thus potentially unreliable information – in particular the reception time of the firing packets of its phase neighbors. The different kinds of (reciprocal) phase shifts are depicted in Figure 6.1 for the sake of clarity.

This stale information problem for single-hop topologies was first described by Degesys et al. in [50]. Patel et al. further analyzed this problem for single-hop topologies in [141]. They also developed the DESYNC-STALE protocol, which – with regard to the jump size parameter

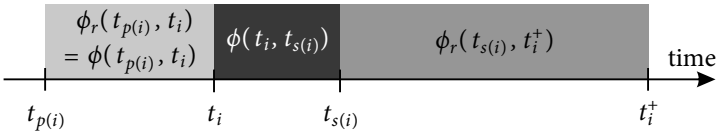


Figure 6.1: Diagram of the different kinds of (reciprocal) phase shifts.

α – is more robust towards stale information than the DESYNC protocol (cf. [141]). The main idea to omit at least the stale information about the successor node is that each node shifts the calculation of its next time of firing.

In particular, node $i \in N$ calculates its next time of firing t_i^+ not immediately after the transmission of its own firing packet at time t_i, but instead immediately after the reception of the first subsequent firing packet of its successor $s(i)$ at time $t_{s(i)}$. As a result, the phase shift $\phi(t_i, t_{s(i)})$ towards this successor is now based on this very current value $t_{s(i)}$ (cf. Figure 6.1). Hence, node i now applies more recent data. Even Eq. (4.14) may remain the same to compute the next time of firing t_i^+. Just the moment when the next time of firing is calculated by node i is shifted from t_i to $t_{s(i)}$. This implementation is already described in Section 5.9 in Listing 5.2 on Page 106 and in Listing 5.3 on Page 107, respectively.

However, node $i \in N$ cannot predict if there (once again) will be a successor transmitting a firing packet. Therefore, by reasons of precaution, node i has to schedule its next time of firing independently from any received firings as $t_i^+ = t_i + T$ immediately after the transmission of its firing message at time t_i (cf. Line 9 in Listing 5.2). However, if node i is receiving the firing of a successively transmitting node, the precautionary scheduled time of its next firing t_i^+ must be replaced by an updated next time of firing according to Eq. (4.14) and Eq. (5.14), respectively (cf. Lines 7 and 8 in Listing 5.3). Once again, just the time when this calculation has to be performed by node i changes from t_i to $t_{s(i)}$. Nevertheless, the information about the predecessor remains stale. According to [50, 141], this stale information does not affect the functionality of the algorithm (for single-hop topologies), but slows down its convergence rate.

The impact of the point in time, *when* a node estimates its next time of firing, is analyzed in [95][2]. Here, a real-world testbed was installed representing a single-hop topology C_{4S} of four nodes plus sniffer (cf. Section 2.4.5). On the one hand, each node was set up to adjust its next time of firing immediately after broadcasting its current firing packet. On the other hand, each node was additionally set up to re-adjust its next time of firing when the node received the firing packet (after its own firing). As a result, the stable state of (perfect) desynchrony is achieved faster in the second case, i.e., if each node re-adjusts its next time of firing after the reception of its successor's firing packet. As this particular packet is sent by the node's successor (at least in single-hop topologies), the most current information then is locally available for the receiving node to compute its adjustment function according to Eq. (4.13b). We will revisit this experiment in more detail in Section 7.3.

[2] Bachelor thesis conducted in conjunction with this work.

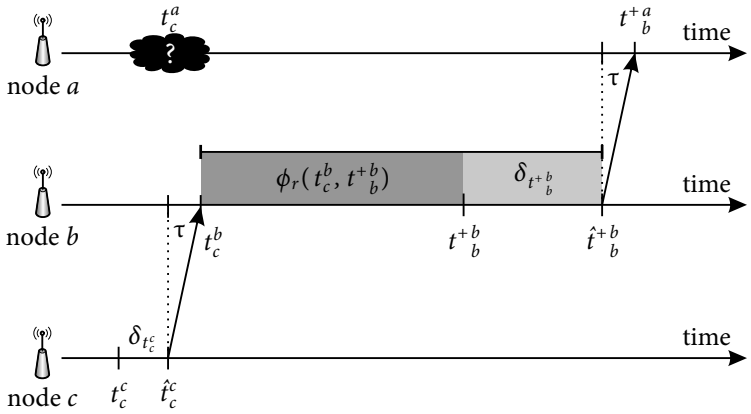


Figure 6.2: The information delay, when node a wants to gain knowledge about the firing time of its two-hop neighbor $c \in N_2(a)$ by means of its one-hop neighbor $b \in N_1(a)$.

6.2.2 Multi-Hop Topology

For a variety of multi-hop topologies, it may be sufficient that each node $i \in N$ just shifts the calculation of its next time of firing t_i^+ from t_i to $t_{s(i)}$. However, the impact of the stale information problem is intensified in multi-hop topologies due to the hidden terminal problem: In this regard, each node i additionally has to take care of its two-hop neighbors $N_2(i)$ (cf. Section 5.3.1). Realizing the phase shift propagation approach from Section 5.3, node i gains information about its two-hop neighbor $k \in N_2(i)$ just in cooperation with the corresponding one-hop neighbor $j \in N_1(i)$ with $k \in N_1(j)$. In fact, this information flow is further delayed by at least the phase shift $\phi(t_k, t_j) > 0$ between j and k. As a result, when node i calculates the firing times of its two-hop neighbors according to Eq. (5.11), this computation is based on stale information. Recalling the sample scenario from Section 5.5.1, node a gains knowledge about the firing time of its two-hop neighbor node $c \in N_2(a)$ just by means of its one-hop neighbor node $b \in N_1(a)$ with node $c \in N_1(b)$ after at least $\phi_r(t_c^b, t^{+b}_b) > 0$. A scenario exemplifying this particular delay $\phi_r(t_c^b, t^{+b}_b)$ is depicted in Figure 6.2.

For a better illustration of the impact of stale information in multi-hop topologies, we use the rather small but manageable dumbbell topology D_7 as shown in Figure 6.3: The network consists of seven nodes $N_{D_7} = \{a, \ldots, f, h\}$. Noteworthy, this topology D_7 contains two cyclic (and complete) subgraphs C_{3l} and C_{3r} with $N_{C_{3l}} = \{a, b, c\}$ and $N_{C_{3r}} = \{d, e, f\}$. Both subgraphs are connected just by means of the *helping* node h. As a result, without node h there would remain two disjoint connected components, and thus two individual networks (cf. Definition 2.17).

For our simulation we assume idealized conditions, i.e., all communication links are bidirectional and reliable, not any node will fail, and there is no clock drift. As protocol parameter, we set period $T = 1\,000\,000$ µs and damping factor $\alpha = 0.95$. Next, let the nodes of the complete subgraphs C_{3l} and C_{3r} start node by node first. In compliance with Lemma 2.1, both subgraphs are unaware of each other and thus will desynchronize independently as long as node h is not running. When node h joins the network after a while, in the first step it gathers

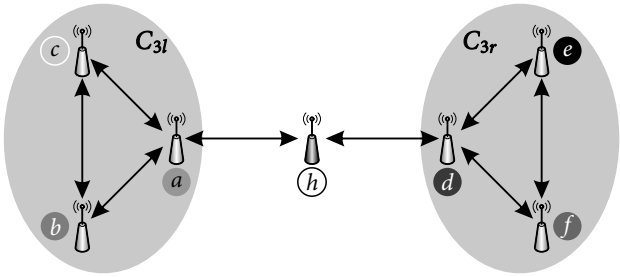


Figure 6.3: The topology D_7 consists of the set $N = \{a, \ldots, f, h\}$ of nodes with bidirectional links.

knowledge about the yet desynchronized subgraphs C_{3l} and C_{3r}. Next, the firing packets of node h contain information about its one-hop neighbors, namely node a and node d. As a result, node h will connect both subgraphs C_{3l} and C_{3r} with its first firing packet. The system eventually forms the topology D_7.

An extract (about 110 periods) of the outcome of a simulation run for this scenario is shown in Figure 6.4: From the point of view (POV) of node h, the neighborhood switches between two distinct phase values. This undesired behavior is due to the stale information about the two-hop neighbors within this multi-hop topology. In contrast to the idea behind the primitive of desynchronization, the one-hop and two-hop neighbors of node h (i.e, all nodes of C_{3l} and C_{3r}, respectively) seemingly rather diverge than converge. In fact, the time of transmission of each node fluctuates with a constant but individual amplitude. Besides, the phase neighbors of node h are two-hop neighbors. According to Figure 6.4, these phase neighbors are the topmost and the lowermost nodes, namely node f and node b (cf. Figure 6.3).

This fluctuation is caused by the stale information about two-hop neighbors being phase neighbors: Using the scenario above, node d gains knowledge about its two-hop neighbor node a just by means of node h. Indeed, the information flow about the firing time of node a towards node d is delayed by $\phi(t_a, t_h)$, and about the firing time of node d towards node a by $\phi(t_d, t_h)$. As a result, the nodes of subgraph C_{3l} and C_{3r} behave like the teeth of two gear wheels moving back and forth all the time. Obviously, the nodes are not able to overcome this perpetuating situation – even though each node propagates the neighbor information of its one-hop neighbors according to Section 5.5.

6.3 Refractory Threshold

In Section 6.2, we specified the stale information problem with special focus on multi-hop topologies. In this section, we present our approach to solve the stale information problem in multi-hop topologies while still being compliant to the primitive of desynchronization: We start with the introduction of our *refractory threshold* as probabilistic protocol parameter in Section 6.3.1. Next, we briefly analyze the parameter's impact on multi-hop topologies in Section 6.3.2. We complete this section with some related work about this topic in Section 6.3.3.

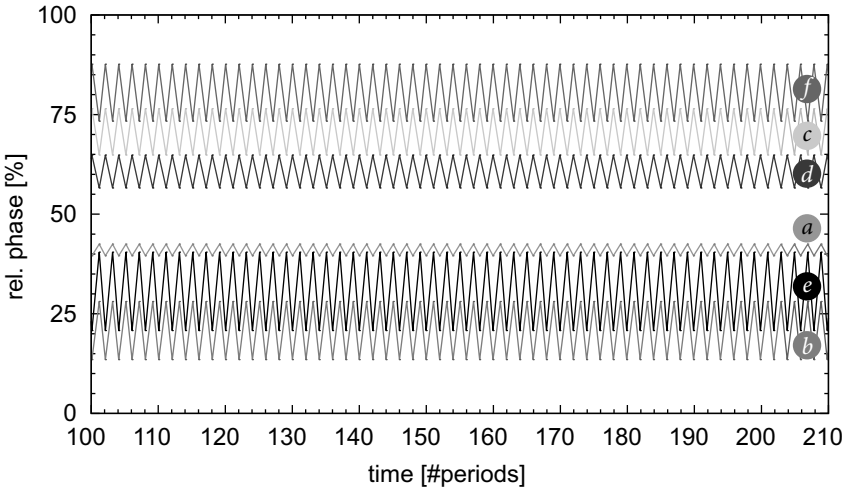


Figure 6.4: Excerpt of a simulation run of topology D_7 from Figure 6.3 ($T = 1\,000\,000\,\mu s$; $\alpha = 0.95$; POV: node h).

6.3.1 Basic Idea

As already mentioned, the stale information problem is inherent to the primitive of desynchronization. Hence, we are not able to avoid the stale information problem at all. Moreover, this problem may compel the nodes of a multi-hop topology to transmit their firing packets in an alternating and never-ending manner as exemplified in Figure 6.4 (cf. Section 6.2.2). Therefore, we want to understand its appearance, and – even more important – we want to reduce its impact on multi-hop topologies. However, the nodes will never desynchronize unassistedly from such an alternating behavior.

To relax the stale information problem and to eventually escape this "vicious circle", our main idea follows the *law of similars*[3] to some extent: The adjustment of a node's next time of firing shall be delayed intentionally. For this purpose, we introduce the *refractory threshold* $\rho \in [0,1]$ as additional protocol parameter. As already stated in [124, 126], we specify the refractory threshold ρ along with a continuous random variable $X_i \in [0,1)$, following the continuous uniform distribution $\mathcal{U}(0,1)$.

As a result, the adjustment function $\varphi_i(N_R(i), t_i)$ and the adjustment factor ε_{t_i}, respectively, will be considered by node $i \in N$ just sporadically – depending on this new probabilistic parameter ρ. According to Eqs. (4.14) and (5.14), node i is setting its next (absolute) time of firing t_i^+ after the reception of its successor $s(i) \in N_1(i) \cup N_2(i)$ at time $t_{s(i)}$ as

$$t_i^+ = \begin{cases} t_i + T & X_i < \rho \\ t_i + T + \alpha \cdot \varepsilon_{t_i} & \text{otherwise} \end{cases}. \tag{6.1}$$

Similar to the jump size parameter α from Section 4.3, the boundaries of the interval of the refractory threshold $\rho \in [0,1]$ are of special importance:

[3] *Similia similibus curentur*, i.e., let like be cured by like.

- Setting $\rho = 0$ lets the node always adjust its next time of firing. The resulting behavior would be identical to that one emerging from the EXTENDED-DESYNC protocol. For simulation purposes, e.g., to compare the EXTENDED-DESYNC with the EXTENDED-DESYNC$^+$ protocol, this setting may be useful.

- Setting $\rho = 1$ forces the node to not use the adjustment function anymore. Consequently, the next time of firing will never be adjusted. As mentioned in Section 4.3, setting the jump size parameter $\alpha = 0$ results in the same behavior. This can be useful when integrating a sniffer (cf. Section 2.4.5), which should act as reference or "referee", as exemplified in Chapter 7.

To some extent, the refractory threshold ρ contradicts the primitive of desynchronization: The adjustment of the next time of firing may be "skipped". However, the refractory threshold as probabilistic parameter enables a node to keep its phase, i.e., to not adjust its next time of firing. This could be one option for the system to eventually reach the stable state of desynchrony: For instance, let node $i \in N$ be phase neighbor of another node $j \in N_1(i) \cup N_2(i)$. According to Lemma 4.8, node j in turn needs not to be the opposing phase neighbor of node i. Moreover, node i and node j both implement the refractory threshold approach according to Eq. (6.1). Assuming, node i is skipping the adjustment of its next time of firing due to the random variable X_i. As a result, the estimation of its next time of firing of node j remains valid regarding its phase shift towards node i. Consequently, the information about node i is still reliable. Since each node decides individually based on its own Pseudo Random Number Generator, this approach seems to cope with the stale information problem adequately. Thus, we will next analyze the impact and benefit of our refractory threshold in the next section.

6.3.2 Impact

To get a first impression, we exemplify the impact of our new refractory threshold ρ on the sample scenario from Figure 6.3. In particular, the two disjoint single-hop topologies C_{3l} and C_{3r} are joined by node h. We use our self-developed simulator EXTDESIMC as introduced in Section 2.3. Again, we set period $T = 1\,000\,000\,\mu s$ and damping factor $\alpha = 0.95$ as protocol parameter. The other parameters remain unchanged.

In contrast to the scenario described in Section 6.2.2, each node i now calculates its next time of firing t_i^+ according to Eq. (6.1). Moreover, each node uses $\rho = 0.25$ as refractory threshold. This means that each node keeps its phase at every fourth firing on average. If a node is keeping its phase, all nearby receiving nodes estimate their next times of firing on more reliable, i.e., less stale information. As a result, the refractory threshold supports the network to converge. Eventually, the system reaches the stable state of desynchrony, i.e., the time of transmission of the nodes is not fluctuating any more. This behavior is depicted in Figure 6.5 presenting the same excerpt as shown in Figure 6.4 – just implementing the new refractory threshold ρ. Noteworthy, the ordering of the nodes in Figure 6.5 is identical to that one depicted in Figure 6.4 – at least from the view of node h.

Certainly, this sample scenario cannot replace a deep analysis. Thus, we will analyze the refractory threshold in more detail in Section 7.4. Nevertheless, this first impression indicates the capability of the refractory threshold.

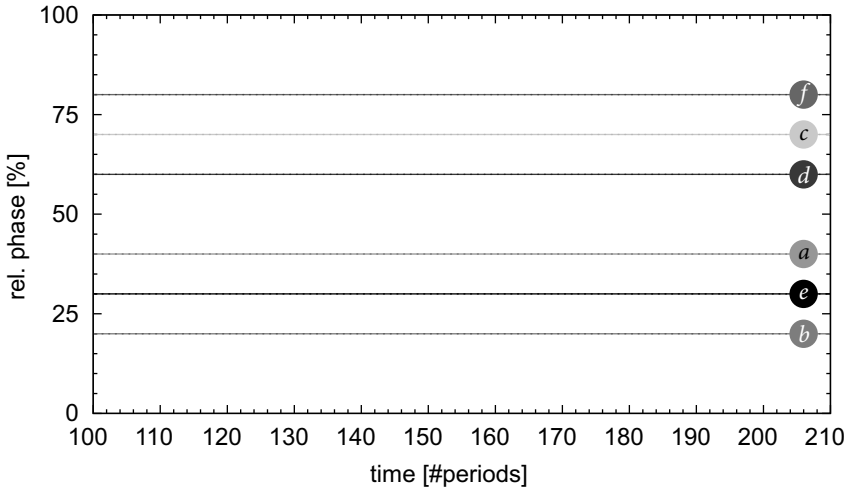


Figure 6.5: Excerpt of the outcome of a simulation run of topology D_7 utilizing the refractory threshold ρ ($T = 1\,000\,000$ µs; $\alpha = 0.95$; $\rho = 0.25$; POV: node h).

6.3.3 Related Work

We introduced in Section 6.3.1 the refractory threshold ρ to cope with outdated and thus unreliable data. Reworded, consciously we make data of a node obsolete for one period T to obtain more reliable data. Our approach is quite similar to the *refractory period*, i.e., a node does not register any incoming firing packets for this certain amount of time. The refractory period and the refractory threshold, respectively, are both biologically inspired. As in nature, such an "resisting time" is essential to prevent harmful oscillations, e.g., on neurons.

The refractory period was suggested in [51, 99] to synchronize (strongly) pulse-coupled oscillators: If an oscillator receives the firing of a neighbor within this refractory period, the receiving oscillator must not further process this incoming firing. This means that the particular phase shift between sender and receiver is less than the refractory period, and thus will not be considered by the receiving oscillator. As a consequence, the receiver temporarily does not adjust its next time of firing. Since our goal is to desynchronize the firings of nodes, this synchronization process is contrary to our objective. Moreover, the utilization of the refractory period depends on the interval between a couple of nodes. In contrast, our refractory threshold ρ is devised as a constant parameter – similar to the damping factor α.

However, [89] as well as [51] suppose the refractory period to be a serviceable parameter for (strongly) pulse-coupled oscillators. This should improve the convergence behavior and the convergence rate, in particular. Indeed, the resulting model may get more complex and thus more difficult to be calculated. Although the refractory threshold approach slightly differs from the refractory period approach, our experience so far in Section 6.3.2 confirms these assumptions. Nevertheless, we will have to analyze the convergence behavior utilizing the refractory threshold in more detail in Section 7.4. However, we first make a short excursion about the generation of the essential random variable X_i in the next section

6.4 Excursion: Pseudo Random Number Generator (PRNG)

We introduced in our EXTENDED-DESYNC$^+$ protocol the refractory threshold ρ as probabilistic parameter. For an implementation in our simulator EXTDESIMC as well as in our real-world testbeds, we are in need of a *Pseudo Random Number Generator* (PRNG). Therefore, we first describe the implementation of a Pseudo Random Number Generator for our sensor node framework in Section 6.4.1. In Section 6.4.2, we present implementation details for the Pseudo Random Number Generator of our simulator EXTDESIMC.

6.4.1 PRNG Implementation for our Sensor Node Framework

As already mentioned, we rely on a Pseudo Random Number Generator due to several reasons (cf. Sections 2.2.2 and 5.8). However, to implement the refractory threshold approach from Section 6.3.1 the availability of such a Pseudo Random Number Generator now becomes mandatory, and is of particular importance: In fact, if the "random" variable X_i from Eq. (6.1) is chosen inappropriately, each node would skip the adjustment of its next time of firing – in worst case simultaneously. This effect is not desired and shall be prevented.

There are several approaches and techniques on how to implement a Pseudo Random Number Generator (cf. textbooks on random number generation like [71, 137]). These generating methods differ in complexity, reproducibility, and contingency of the generated random numbers (cf. [1, 8]). For our purposes, the reproducibility of the used generator is important. In particular, we want to keep the system behavior rather controllable and manageable than to generate perfectly randomized numbers – especially for our simulator EXTDESIMC (cf. Section 2.3.2).

Furthermore, the computational effort of a node to generate such a random number has to be considered as well. Due to the fact that the computational power of sensor nodes deployed in our real-world testbeds is quite limited (cf. Section 2.4), we choose a *Linear Congruential Pseudo Random Number Generator* [8, 105] which is solely based on simple calculations. For instance, this kind of a PRNG equals Java's default implementation of a Pseudo Random Number Generator (cf. `java.util.Random`). The basis of calculation of the Linear Congruential PRNG is outlined in Listing 6.1.

```
1 seed = (seed * multiplier + summand) mod 2^precision
```

Listing 6.1: Pseudo-code of Java's default PRNG implementation.

For the used software framework of our sensor nodes, there already exists a simple but feasible implementation of such a Linear Congruential Pseudo Random Number Generator, written in C. The functions of the SMARTOS compliant library are shown in Listing 6.2: Besides the declaration of necessary constants and variables in Lines 2 to 4, this software component just offers three functions as follows: First of all, there is the initialization routine in Lines 6 to 8 to prepare the generation of random numbers with an (application) specific initial value (*seed*). For instance, the node ID may be used as seed here (cf. Listing 6.3). Next, there are two functions left to return a random number: The first function in Lines 10 to 12 returns the random number as `uint16_t`. The second one in Lines 14 to 16 calls the first one, but returns the resulting random number as `char` value. The latter function complies well with Java's default implementation in Listing 6.1.

```
// PRNG functionality of SmartOS
static const uint_16 factor = 277;
static const uint_16 add = 2455;
static uint_16 seed;
// Initialization
void prng_init(uint_16 _seed) {
  seed = _seed;
}
// Return of pseudo random word
uint_16 prng_getWord(void) {
  return seed = (seed * factor) + add;
}
// Return of pseudo random byte
unsigned char prng_getByte(void) {
  return prng_getWord() >> 4;
}
```

Listing 6.2: The SmartOS-compliant implementation of a PRNG.

In our real-world testbeds, each node has to generate random numbers. Consequently, each node runs its own Pseudo Random Number Generator to get these numbers. Besides, each sensor node features a unique ID for identification and recognition (cf. Sections 2.4 and 5.6). As already mentioned in Section 5.6, the node ID also is stored in the infomem of the assembled microcontroller MSP430 (cf. Section 2.4). The SmartOS operating system provides the specific function `infomem_getNodeID()` to read this special data from the infomem. Therefore, we will use this unique identifier as seed to initialize the Pseudo Random Number Generator as presented in Listing 6.3 for the implementation of our real-world testbeds (cf. Listing 5.1).

```
prng_init(infomem_getNodeID());
```

Listing 6.3: Using the node ID as seed for the PRNG.

6.4.2 PRNG Implementation for our Simulation Framework

For our simulations, we make use of the Pseudo Random Number Generator provided by the Java programming language as described in Section 6.4.1 and Listing 6.1. In fact, there is no need that each simulated sensor node operates its own instance of a Pseudo Random Number Generator. However, the behavior of a node of the real-world testbed would be reflected more precisely by node-specific instances. For the sake of efficiency and to get comparable simulation results, we instantiate just one "global" Pseudo Random Number Generator per simulation. Of course, each node within the simulation gets access to this instance. Consequently, there is just one "global" seed per simulation scenario to initiate the generation of random numbers. This seed can be specified within the simulation configuration (cf. Section 2.3). Initially, we use the same default seed, namely 8 690 401 185 424 030, for our simulation tool (implemented in Java) as well as for our script (implemented in Perl) to generate simulation models (cf. Sections 2.3.2 and 2.3.4). Certainly, the seed of the simulator extDeSIMc as well as of the script may be adjusted independently of each other. Please note, since our embedded software expects the seed as `uint16_t` value for initialization according to

Listing 6.2, the default seed used for the Java and Perl implementations is too big to be reused for our sensor node framework from Section 2.4 (cf. Listing 6.4). This is an additional argument in favor of utilizing the node ID as seed for the real-world testbeds as described above.

6.5 Summary

In Chapter 5, we described our self-organizing MAC protocol EXTENDED-DESYNC. This protocol is quite flexible and applicable for arbitrary multi-hop topologies. However, we discovered a certain scenario in which the EXTENDED-DESYNC protocol was not able to desynchronize the system. The main reason for this behavior is the stale information problem as introduced in Section 6.2.

As a solution, we introduced the *refractory threshold* as probabilistic parameter for our EXTENDED-DESYNC$^+$ protocol (for multi-hop topologies) in Section 6.3: Depending on this threshold, a node will skip the adjustment of its next time of firing – or not. These individual decisions help the system to eventually reach the stable state of desynchrony. The usage of this probabilistic parameter now strictly requires the application of a Pseudo Random Number Generator. Therefore, we inserted a brief excursion about the particular implementation and utilization of a Pseudo Random Number Generator for our simulator EXTDESIMC and for our real-world testbeds in Section 6.4.

Next, we present the algorithm to calculate the next time of firing of a node $i \in N$ of our EXTENDED-DESYNC$^+$ protocol. Since the event on sending a firing packet by node i remains unchanged in relation to our EXTENDED-DESYNC protocol as shown in Listing 5.3, we just inspect the event on receiving a firing packet from a neighbor node by node i in Listing 6.4. Here, node $i \in N$ is receiving at least one firing packet from a neighbor node before the expiration of its firing timer (cf. Lines 1 to 12): As already described in Section 5.9, the received firing packet may be the first one after node i's own transmission – or not. In comparison to the algorithm of the EXTENDED-DESYNC protocol presented in Listing 5.3, there is only one adjustment made. In Line 6 in Listing 6.4, we apply a ternary operator (namely `? :`) to implemented the conditional expression of Eq. (6.1). If the refractory threshold ρ is less a random value, then the adjustment factor $\varepsilon_{t_i^i}$ is set according to Eq. (5.14a), else it is set to 0.

```
1  // upon receiving a firing packet
2  if (isFirstPacketReceived()) {
3    setFirstPacketReceived(false);
4    t^i_{s(i)} = now();
5    φ(t^i_i, t^i_{s(i)}) = (t^i_{s(i)} − t^i_i);
6    ε_{t^i_i} = (ρ < Random.nextDouble()) ? (φ(t^i_i, t^i_{s(i)}) − φ_r(t^i_{p(i)}, t^i_i))/2 : 0;
7    t^{+i}_i = t^i_i + T + α · ε_{t^i_i}; // overrides the next time of firing
8    setFiringTimer(t^{+i}_i);
9    } else {
10     t^i_{p(i)} = now();
11     φ_r(t^i_{p(i)}, t^{+i}_i) = (t^{+i}_i − t^i_{p(i)});
12 }
```

Listing 6.4: The calculation of the next time of firing t^{+i}_i of node $i \in N$ integrating the refractory threshold ρ as implemented in our EXTENDED-DESYNC$^+$ protocol.

So far, we just did exemplify the impact of this probabilistic parameter in Section 6.3.2. In addition, no universal value for the refractory threshold has been elaborated so far. We just used a certain value for this probabilistic parameter for the purpose of illustration. A deeper analysis of this threshold is missing. For this reason, we refer to the next chapter, and to Section 7.4 in particular.

Part III

Evaluation

> Was macht die Zeit, wenn sie vergeht?
>
> *Albert Einstein*

Abstract

In the previous part, we introduced the underlying primitive of desynchronization and presented our communication protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$. With this current part we conclude this work. This means, we will analyze our protocols, discuss the results of our experiments and simulations, and finally summarize the main results of our work.
In particular, Chapter 7 evaluates our self-organizing MAC protocols. For this purpose, we use the mechanisms simulation and experiments as described in Section 2.2. This means that we will analyze the protocol handling as well as the basic protocol parameters, namely jump size parameter α and refractory threshold ρ. Moreover, we will also oppose our self-organizing protocols to our requirements from Section 1.2 within this chapter. The outcome of this analysis is discussed in Chapter 8: In this chapter, we give a short outlook to future work. Furthermore, we also present some (network) services to be realized as add-ons for our self-organizing protocols. Finally, Chapter 9 sums up the key aspects, and concludes this work.

Chapter 7

Analysis

Abstract

In this chapter we analyze several aspects of our self-organizing protocols EXTENDED-DESYNC from Chapter 5 and EXTENDED-DESYNC$^+$ from Chapter 6, respectively. Especially, we will check whether our requirements from Section 1.2 are met. In this regard, we also will motivate the protocol-specific parameters, namely the jump size parameter α and the refractory threshold ρ. For this purpose, various techniques, namely simulation, experiment, and analytical solution, are feasible (cf. Section 2.2). Indeed, for all our analyses we just did set up experiments with real-world testbeds (cf. Section 2.2.1) as well as simulation runs of adequate models with our self-developed simulator EXTDESIMC (cf. Section 2.2.2).

Since a simulation model always just abstracts the real-world, we first validate our simulation model in Section 7.1. In Section 7.2, we demonstrate the (undesired but complicating) fact that even small changes in the system setup may result in a totally different outcome. Next, we analyze the distinctive protocol parameters of our MAC protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$. In particular, we examine the jump size parameter α in Section 7.3 and the refractory threshold ρ in Section 7.4. Our requirements from Section 1.2 are checked in Section 7.5: This means that we will verify applicability, scalability, and robustness against environmental perturbations of our self-organizing communication protocols to meet our demands. Finally, Section 7.6 concludes this chapter.

7.1 Simulation Model Validation

In Section 2.3, we described our self-developed simulator EXTDESIMC in detail. However, any conclusion drawn from simulation runs is at most as reliable as permitted by the underlying simulation model. Therefore, the validation of a simulation model is an essential procedure to get sound results and to draw reliable conclusions. For this reason, this section describes the validation process of our simulation model against the corresponding real-world scenario.

7.1.1 Object of Investigation

It is always a hard task to create a simulation model of a physical system when the creation process is exclusively based on certain technical characteristics of this system. Hence, the sole values from the data sheets of the assembled hardware components of our real-world testbeds are not sufficient to specify our simulation model properly. We illustrate this proposition by

the phenomenon *clock drift*, which is inherent to networked systems but difficult to emulate in detail[1]:

1. First of all, the data sheet values for an external quartz crystal, which drives the node's microcontroller, are not sufficient at all: Some of its parameters depend on specific environmental conditions. For instance, the frequency tolerance of the crystal quartz model HC49/3H from IQD, which is assembled on the SNoW[5] sensor node, was specified at 25 °C ± 2 °C (cf. [88]). In general, these values are not specified in correlation to other environmental conditions, e.g., at 10 °C. Furthermore, it also is not likely that the system environment always fulfills these specific conditions.

2. Next, the frequency stability of an external quartz crystal driving the node's microcontroller is influenced by several factors like aging, temperature, and humidity (cf. Definition 2.6). This environmental impact is significant and not negligible, especially for outdoor deployments where parts of the network could be exposed to direct sunlight, whereas other parts of the network may lay in the shade. For a very accurate and fine grained model, all these factors would have to be taken into account. However, even though it may be possible to determine all physio-chemical properties of the surrounding for each quartz crystal by placing adequate sensors nearby each crystal, it is quite difficult to determine the aging of a crystal quartz precisely (cf. [136]).

3. Moreover, some properties in the data sheet are not specified as single value but as range of values instead. For instance, the effect of aging on the frequency stability of the IQD crystal quartz HC49/3H model is specified from −5 ppm up to +5 ppm per year (cf. [88]). Setting this parameter for a simulation model accordingly is hardly possible[2].

4. In addition, clock drift is just a statement between two clocks. As our self-organizing system usually does not provide a global reference clock (cf. Section 1.2), the clock drift here is always specified between pairs of clocks. As a consequence, for an accurate simulation model not only information about the frequency stability of each crystal quartz assembled but also information about the clock drift between each pair of interacting nodes[3] would be required. Depending on the network size, network density, and the network topology, this is a quite difficult task.

5. Since all digital systems use time not as continuum but in discretization, inaccuracies in this regard are rather inevitable. Indeed, techniques to minimize these inaccuracies yet exist. Furthermore, methods to get just symmetrical errors on timestamping are also available, for instance as described by Baunach in [19]. Nevertheless, there remains some inaccuracy to a certain extent.

6. Finally, there still may be further but arbitrary and non-deterministic delays in the individual timing of a node. Such delays may be caused by hardware and/or software

[1] Nevertheless, our simulation model allows the definition of a constant clock drift rate for each node.

[2] Measuring the crystal's inaccuracy (e.g., by an oscilloscope) just determines the overall inaccuracy, but not the causing factors, like aging, in isolation.

[3] Noteworthy, in fully connected networks the number of pairs grows quadratically with the number of nodes.

issues. For instance, entering a subroutine (e.g., an ISR) due to an (external) event (causing an IRQ) supports this effect. Such sporadic effects are hard to simulate, but could be emulated using probabilistic elements during simulation.

Hence, we were not able to set up an accurate model just from data sheet values at reasonable expense. Instead, we had to use measurements taken from real-world testbeds as additional data basis to refine the corresponding simulation model. For this reason, we compare the behavior of a real-world testbed against the outcome of the corresponding simulation model iteratively – until the model matched the behavior of the testbed.

Moreover, assuming that it would be possible to model every property of the real-world testbed and its surrounding, this high level of detail may be undesired: The size as well as the complexity of the simulation model would grow tremendously. This additionally affects the time required for a single simulation run. Accompanied is the difficulty to efficiently handle such a huge model. As a consequence, we are trying to keep the simulation model as accurate as necessary but also as simple as possible (cf. Section 2.3.3). Therefore, we do not aim for a simulation model as exact replication of physical effects but as a valid representative of scenarios related to the primitive of desynchronization by tolerating slight but comprehensible differences. Indeed, we rather want to observe and monitor the system behavior, i.e., the convergence behavior of the nodes in particular (cf. Section 2.3).

7.1.2 Expectation

After the development phase, in which the simulation model was iteratively implemented and improved according to the oucome of the underlying real-world testbed, we expect just little deviations. Since there are inaccuracies in time measurement and timing (cf. Section 7.1.1) and since our focus mainly is on the convergence behavior of the system (cf. Section 1.2), we do not rely on a very fine-grained simulation model. This means that we do not insist on the exact replication of physical effects. This is the reason why we could not expect an identical output but little deviations for a simulation scenario. These deviations in general may differ in

Convergence Rate One variant (e.g., simulation) may converge considerably faster than the other (e.g., testbed).

Set of Neighbors In the stable state of (perfect) desynchrony, the transmission order of the nodes may differ, i.e., reflected by a permutation of the node's neighborhood.

Phase Shift The (reciprocal) phase shift between a pair of nodes of one variant (e.g., simulation) may differ from the (reciprocal) phase shift of the same pair of nodes of the other variant (e.g., testbed).

If the system reaches the stable state of (perfect) desynchrony, we just expect permutations of neighbors. This permutation may appear as shifted transmission times or as a different order of neighbor nodes. At least, the phase shift between successor and predecessor is expected to be identical. Since we do not consider identical start up times but an identical start up order for simulation and real-world testbed, we have to run the same simulation model for several times to smooth out potential deviations.

7.1.3 Procedure

The wireless communication of sensor nodes is influenced by environmental conditions. For instance, the RSSI value of the RF unit (cf. Section 2.2.1) is influenced by moving people or the opening and closing of a door. Since our simulation framework extDeSIMc allows the generation of fully observable as well as reproducible scenarios, it is easier for us to refine the simulation model according to the underlying real-world testbed than to modify the corresponding testbed. Therefore, we always run the real-world testbed first and – according to the recorded data of the testbed – we develop and adjust the simulation model then.

Neither is it possible to set up all potential scenarios, nor is our stock of real sensor nodes unlimited. Hence, within this section we focus on some small, but manageable scenarios of real-world testbeds consisting of up to six SNoW^5 sensor nodes (cf. Section 2.4.1), including active ones as well as sniffers (cf. Section 2.4.5). The validation procedure for each specific scenario was as follows:

V1. First, we installed a particular real-world testbed consisting of SNoW^5 sensor nodes at the laboratories. The objective was to construct a particular topology by skillful and clever arrangement of the sensor nodes.

V2. This testbed ran for a while logging data[4]. As mentioned before in Section 7.1.1, a sensor node not only acts as object of investigation but also as measuring device. This means that each node logs every incoming packet, every firing packet transmitted, and its list of one-hop as well as two-hop neighbors at its time of firing. All these data is provided at the serial communication interface of the node, e.g., to be stored by a terminal program at a PC.[5]

V3. According to the recorded data, we developed a simulation model primarily emulating the topology and the start up order (with randomized start up times) of the nodes. Certainly, we set up identical protocol parameters (i.e., jump size parameter α and refractory threshold ρ) for both, simulation and testbed. Due to potential effects like roundings, interferences, and clock drifts, each simulation run was performed at least ten times to compensate such deviations.

V4. Next, we compared the outcome of the particular testbed to the outcome of the corresponding simulation run mainly by visual control.

V5. If the simulation could not reproduce the convergence behavior of the testbed (temporary deviations, e.g., due to moving people, may be tolerable), the simulation model was adjusted and rerun (see Item V3).

7.1.4 Results

As expected, a few times the outcome of the simulation run differed to the outcome of the real-world testbed even after the development phase of the simulation model. These deviations appeared due to the dynamics of the environment as well as the randomized start up

[4] Depending on the underlying topology, we used one or more sniffers for data logging.

[5] Certainly, using the serial communication interface for the output of the logging information additionally affects the system load of a node (cf. Section 2.4).

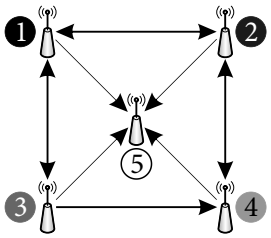


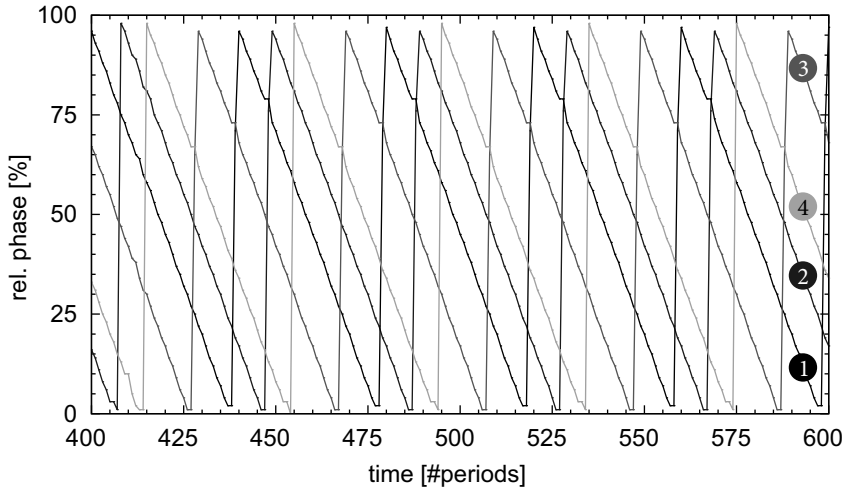
Figure 7.1: The topology A_{4S} consists of the set $N = \{1, \ldots, 5\}$ of nodes with sniffer node 5. Notably, there is a unidirectional link from node 3 to node 4.

times of the nodes. Indeed, we could identify the following configurations promoting the emergence of such deviations:

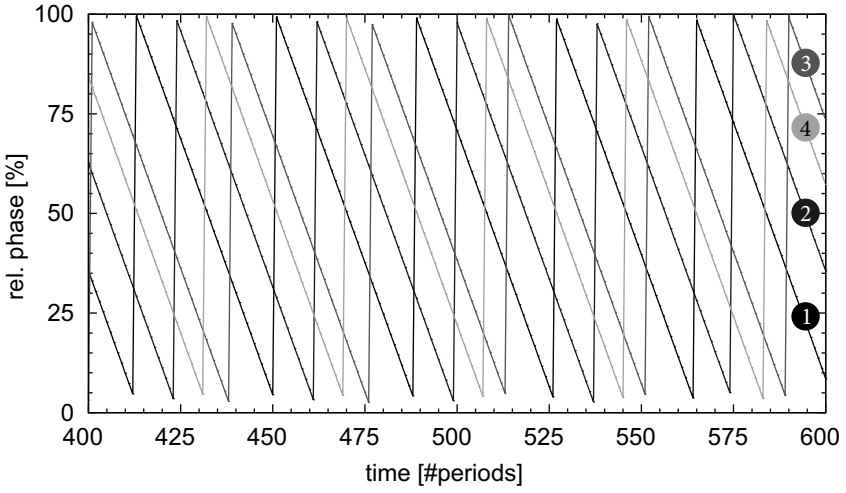
- If the time lag between the start up time of two subsequent nodes within the same constraint graph is too small (i.e., less than the packet transmission time between both nodes), the subsequent node would impair the transmission of other nodes – at least the preceding one. Moreover, the firing packets of these nodes may collide. Finally, these collisions could even destabilize the whole system (cf. Section 5.8).
- If the system has not yet reached the stable state of (perfect) desynchrony but still is in settling mode and tries to desynchronize, nodes starting up and joining the network in the meantime could prolong this settling process. In the worst case, such joining nodes could even destabilize the system – especially in scenarios when all nodes start up within the first period (cf. Section 7.2). This phenomena appears when the temporal distance between the start up time of the joining node is too short (i.e., up to several hundreds of microseconds) to the next time of firing of an already participating node. This means that even Carrier Sense right before firing could not detect such an interruption, e.g., due to the switching time from receive mode to transmit mode of the radio unit (cf. [96]). For instance, the CC1100, which is placed on the SNoW[5], requires 9.6 µs to switch from RX mode to TX mode (cf. [175]).

As an example, we present here the outcome of topology A_{4S}, which is one of the analyzed scenarios (cf. [125]): As depicted in Figure 7.1, the network consists of four active nodes and one sniffer (cf. Section 2.4.5). Noteworthy, the link between node 3 and node 4 is unidirectional, i.e., node 4 receives packets from node 3 but not vice versa. Since node 4 is three hops away from node 3 within this topology, node 3 will gain no knowledge about the presence of node 4 without further exchange of information. For simulation and real-world testbed we set period $T = 500\,000\,\mu s$, damping factor $\alpha = 0.95$, and refractory threshold $\rho = 0$.

For comparison, Figure 7.2 opposes the same extract of the outcome of the real-world testbed in Figure 7.2(a) to the outcome of the simulation run in Figure 7.2(b). Both diagrams in Figure 7.2 show the point of view of the sniffing node 5. The domain axis denotes the time as number of periods since start up of the simulation, the range axis denotes the relative phase in percentage of the period length, i.e., 100 % of a period is identical with 0 % of the subsequent period (cf. Section 2.3). As expected, both graphs look quite similar. The shown convergence behavior seems to be identical.



(a) Excerpt of the outcome of the real-world testbed.



(b) Excerpt of the outcome of the simulation run.

Figure 7.2: Excerpts of the outcome of the real-world testbed and the simulation run of topology A_{4S} (T = 500 000 µs; α = 0.95; ρ = 0; POV: node 5).

Small differences become more obvious in the overlap of both outcomes in Figure 7.3: Due to the limited frequency stability and integer rounding at the real nodes, the abscissas of the outcome of the simulation (using perfect clocks without clock drift) were rescaled by 106 % here. Indeed, the outcome of the simulation run looks more straight-lined than the slightly wavy graph of the real-world testbed. These undulations in the outcome of the real-world testbed are mainly caused by clock drifts and roundings. Nevertheless, the simulation model

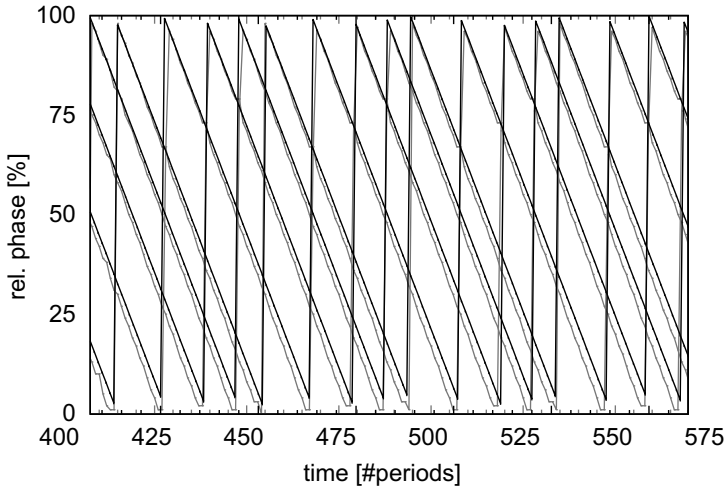


Figure 7.3: Overlap of the outcome of the real-world testbed (gray) from Figure 7.2(a) and the outcome of the simulation run (black) from Figure 7.2(b).

(at least for this particular scenario) can be considered to be valid.

Certainly, this visual comparison is not on a par with an analytical solution. Indeed, in Sections 7.1.1 and 7.1.2, we sketched the potential complexity of a fine-grained simulation model as well as the difficulty to set up a corresponding mathematical (non)linear system (cf. [112]). Since we are mainly interested in the system's convergence behavior, small differences in the outcome as depicted in Figure 7.3 are tolerable for our purposes. Indeed, several dimensioning rules as maximum tolerable deviation for the outcome are arguable. For instance, the difference between simulation and testbed has to be less than a fixed value or less than a particular percentage of packet length or period length. However, these dimensioning rules must not have any notable impact on the system's convergence behavior.

Hence, we specify the following *dimensioning rules* to classify the current state of node:

R1. First, we specify the *stable state of desynchrony* (cf. Definition 4.3) of a node. In compliance with Eq. (4.7), the absolute difference between the value of a node's current adjustment function and the value of its previous adjustment function is less or equal a certain threshold. This threshold depends on the current period T, it shall be in the magnitude of a thousandth part of this period T. For instance, let $T = 1\,000\,000\,\mu s$. Choosing a factor of 1 ‰ then results in a maximum tolerable deviation of $T \cdot 1\,‰ = 1000\,\mu s$.

R2. Next, we specify the *stable state of perfect desynchrony* (cf. Definition 4.3) of a node. For this purpose, it is a mandatory prerequisite that the node already is in the stable state of desynchrony as described in dimensioning rule R1. In addition and in compliance with Eq. (4.8), the absolute value of the previous adjustment function has to be less or equal the half of the threshold from dimensioning rule R1. Continuing the example from dimensioning rule R1, the maximum tolerable deviation of the absolute difference between the previous and the current result of a node's adjustment function is 500 µs.

R3. In any other case, a node is considered to be not in a stable state.

These dimensioning rules are implemented in our simulator extDeSIMc as shown in Listing 7.1: In Line 2, the maximum tolerable deviation is initialized. The function to determine the node's current state of desynchrony is specified in Lines 5 to 27. The `seriesRingBuffer` is used in Lines 9 to 11 to compare the value of the node's past adjustment function with the current one. The node's state of desynchrony is set in Line 24 and Line 26, respectively.

```
1  // setting maximum tolerable deviation
2  this.MAX_TOLERABLE_PHASE_DIFF = T·1‰;
3
4  // determining the node's current state
5  protected DESYNC_STATE getMyDesyncState() {
6    boolean isStable = false;
7    Number lastValue = null;
8    Number currentValue = null;
9    for (int i = 0; i < seriesRingBuffer.getSize(); i++) {
10     lastValue = seriesRingBuffer.get(i);
11     currentValue = seriesRingBuffer.get(i + 1);
12     if (lastValue != null && currentValue != null) {
13       if (Math.abs(currentValue.doubleValue()
14           - lastValue.doubleValue()) > MAX_TOLERABLE_PHASE_DIFF) {
15         isStable = false;
16         break;
17       }
18     }
19     isStable = true;
20   }
21   if (isStable && lastValue != null
22       && (Math.abs(lastValue.doubleValue()) <=
23           (MAX_TOLERABLE_PHASE_DIFF / 2))) {
24     return DESYNC_STATE.PERFECT_DESYNC;
25   } else
26     return (isStable) ? DESYNC_STATE.DESYNC : DESYNC_STATE.OUT_OF_SYNC;
27 }
```

Listing 7.1: Implementation of the dimensioning rules specifying the stable state of (perfect) desynchrony of our simulator extDeSIMc.

Notwithstanding, there seems to be no continuity at the exemplary scenario as depicted in Figure 7.2: The temporal distance between each pair of nodes remains constant, whereas the whole system rotates at constant speed along the period T. In compliance with Definition 4.3 and according to dimensioning rule R2, the stable state of perfect desynchrony will never be reached. This phenomenon is caused by the unidirectional link between node 3 and node 4, which will be further analyzed in Section 7.5.

7.2 Setup Consequences

We selected the midpoint approach in Section 4.3 as basis for our self-organizing MAC protocols due to its simple algorithm. However, the design space of a (self-organizing) multi-hop WSN is quite complex by nature. This complexity is influenced by the following factors:

F1. First of all, the used MAC protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ are configurable in various ways. Hence, its design space is influenced by certain *protocol parameters*, like the damping factor α (cf. Section 7.3) and the refractory threshold ρ (cf. Section 7.4) of the EXTENDED-DESYNC$^+$ protocol.

F2. Next, several parts of the design space are influenced by *network parameters*. This includes typical parameters like network structure, link quality, and network size. Indeed, network changes, like (partial) mobility of the nodes or (arbitrary) topology dynamics, do also affect the system complexity. In fact, these factors are significant for the convergence behavior of the system. In this regard, we stress once more the fact documented in Section 4.2: The deployment and the analysis of a multi-hop topology are much more elaborate than of a single-hop topology.

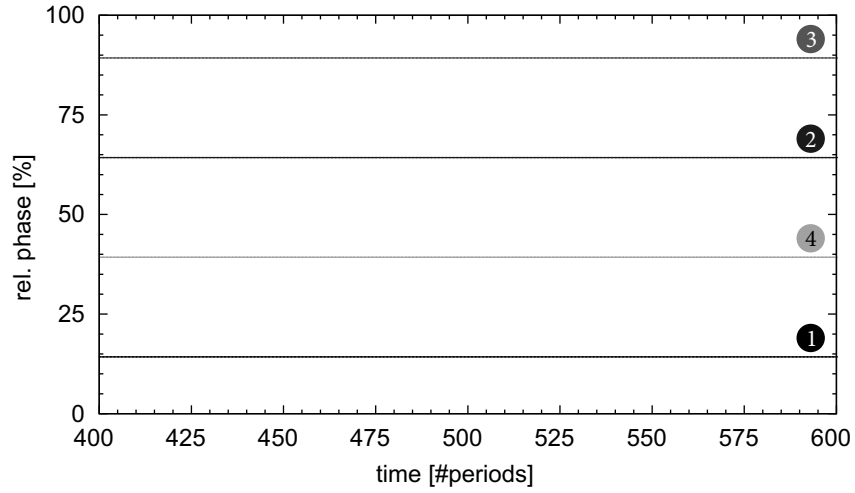
7.2.1 Object of Investigation

During the validation of our simulation model (cf. Section 7.1), we identified another factor which not only further increases the system complexity but also affects the convergence behavior of the system: the *start up order* of the nodes. To illustrate this effect, the sniffer's outcome of one of the simulation models describing the scenario of topology A_{4S} from Section 7.1 is displayed in Figure 7.4(a): Here, the simulated system seems to be in perfect desynchrony – at least from the sniffer's point of view. Notwithstanding, this behavior satisfies our expectations on a self-organizing MAC protocol much more than the outcome from Section 7.1. However, the outcome of this particular simulation as shown in Figure 7.4(a) totally does not match with the observed outcome of a real-world testbed as exemplified in Figure 7.2(a). Moreover, it does not look like the sawtooth wave resulting from simulation runs as exemplified in Figure 7.2(b).

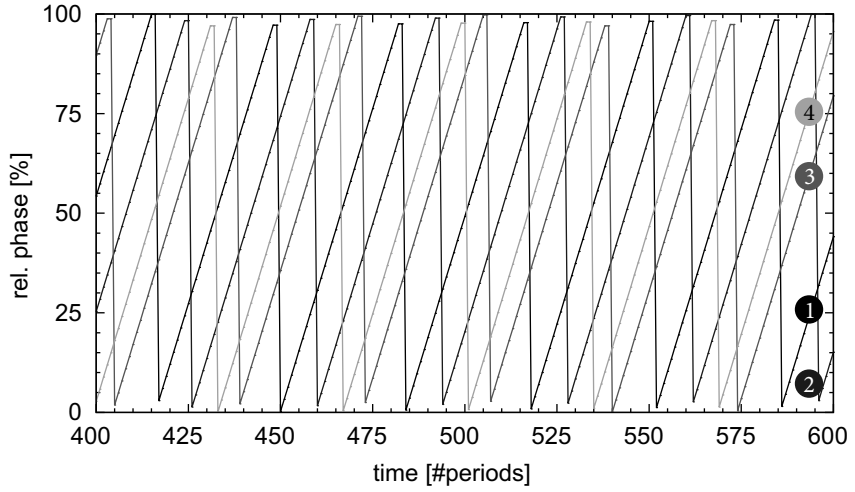
The reason for this difference in outcome was the inadvertent swap of the start up time of node 2 and node 4 in the simulation model. The start up order in this particular simulation model was $(2, 1, 4, 3)$, but should be $(4, 1, 2, 3)$ to correspond with the start up order of the underlying real-world testbed. Therefore, the initial *start up order* of the participating nodes predefines the order of the nodes along the period in a certain way. Moreover, this order obviously affects the convergence behavior of the system. Indeed, a swap of the start up time of nodes results in a different ordering of the nodes but not necessarily in a different convergence behavior as illustrated in Figure 7.4(b): Here, the start up time of node 1 and node 4 in the simulation model was swapped. This means that the start up order of the simulation model was $(1, 4, 2, 3)$ in contrast to the testbed's order $(4, 1, 2, 3)$. Seemingly, the convergence behavior of simulation and testbed are identical. However, the lines of the firing graph in Figure 7.2(b) are going from top left to bottom right, whereas the lines of the firing graph in Figure 7.4(b) are going from bottom left to top right.

7.2.2 Expectation

Our introductory example from Section 7.2.1 already provides two different convergence behaviors – just because of a different start up order of the nodes. For this reason, we could expect several potential convergence results solely based on another start up order and start



(a) Excerpt of the outcome of the simulation run of topology A_{4S}. In comparison to Figure 7.2(b), just the start up time of node 2 and node 4 was swapped here, i.e., the start up order is $(2, 1, 4, 3)$.



(b) Excerpt of the outcome of the simulation run of topology A_{4S}. In comparison to Figure 7.2(b), just the start up time of node 1 and node 4 was swapped here, i.e., the start up order is $(1, 4, 2, 3)$.

Figure 7.4: Excerpts of the outcome of simulation runs of topology A_{4S} with swapped starting times (T = 500 000 µs; α = 0.95; ρ = 0; POV: node 5).

up times in addition. The results could vary from stable state of (perfect) desynchrony to fluctuations and divergence to even chaotic behavior.

As a consequence of the results in Section 7.1.4, the particular start up time of a node impacts the system behavior in a certain way. Thus, it could make a difference, whether a

Start up order	Legend
$(4,1,2,3)$	initial situation, no swap
$(1,4,2,3)$	swap node 4 for node 1
$(2,1,4,3)$	swap node 4 for node 2
$(3,1,2,4)$	swap node 4 for node 3
$(4,2,1,3)$	swap node 1 for node 2
$(4,3,2,1)$	swap node 1 for node 3
$(4,1,3,2)$	swap node 2 for node 3
$(1,2,3,4)$	for comparison purposes

Table 7.1: List of simulated swaps in start up order with $(4,1,2,3)$ as starting point.

node starts for instance at time $0 \cdot T$, $0.5 \cdot T$, or at time $1 \cdot T$. Indeed, we are not able to simulate all potential start up times for each node – not even for that small scenario described in Section 7.1 due to limitations in the available execution time. Thus, we have to focus on the start up order in contrast. Consequently, using different start up orders let us expect to observe additional convergence behaviors, but we do not expect to perceive all potential convergence behaviors, like fluctuations. For instance, we will not simulate the special case that all nodes are starting up at the very same time within this section.

Moreover, the start up order has an impact on the complexity of the system. Hence, we need an appropriate specification on how to get comparable results, i.e., the start up order has to be normalized: This specification should assign certain restrictions and requirements to keep the setup of the simulation manageable. Against this background, we further expect initial indications and recommendations for a specification to model the start up time (and thus the start up order) of the nodes in an efficient and comparable way.

7.2.3 Procedure

As already mentioned, we inadvertently swapped the starting times of the validation scenario for topology A_{4S} from Section 7.1.3. Consequently, the start up order of these two nodes also did swap. Therefore, we further modified the corresponding simulation model of the validation scenario: In particular, we took the start up order of the real-world testbed $(4,1,2,3)$ as starting point and modeled all potential swaps of starting times for two of the four active nodes (i.e., node 1 to node 4) as listed in Table 7.1. For the purpose of comparison, we also simulated the start up order $(1,2,3,4)$. For all these simulation models, we set period $T = 500\,000\,\mu s$, damping factor $\alpha = 0.95$, and refractory threshold $\rho = 0$.

Since this small number of simulation models seems not to capture all potential convergence behaviors, we wanted to simulate additional models of this topology using the same settings. For this reason, we have to modify both, start up times as well as start up order. Indeed, the start up times of the nodes in the hitherto existing models are quite scattered, e.g., the temporal distance between the first and the last starting node is about 30 s = 60 periods. This is quite a long time for our simulation model being accurate to a microsecond. However, as already mentioned, we are not able to simulate all potential start up times – and certainly not for such a long time span. Therefore, we intended to use randomized start up times as

Order	Node 1	Node 2	Node 3	Node 4	Outcome
(4, 1, 2, 3)	3 539 940 µs	10 229 500 µs	30 500 564 µs	368 500 µs	Fig. 7.2(b)
(1, 4, 2, 3)	368 500 µs	10 229 500 µs	30 500 564 µs	3 539 940 µs	Fig. 7.4(b)
(2, 1, 4, 3)	3 539 940 µs	368 500 µs	30 500 564 µs	10 229 500 µs	Fig. 7.4(a)
(3, 1, 2, 4)	3 539 940 µs	10 229 500 µs	368 500 µs	30 500 564 µs	Fig. 7.4(b)
(4, 2, 1, 3)	10 229 500 µs	3 539 940 µs	30 500 564 µs	368 500 µs	Fig. 7.4(a)
(4, 3, 2, 1)	30 500 564 µs	10 229 500 µs	3 539 940 µs	368 500 µs	Fig. 7.4(a)
(4, 1, 3, 2)	3 539 940 µs	30 500 564 µs	10 229 500 µs	368 500 µs	Fig. 7.4(b)
(1, 2, 3, 4)	368 500 µs	3 539 940 µs	10 229 500 µs	30 500 564 µs	Fig. 7.4(a)

Table 7.2: Starting times of the nodes from topology A_{4S} and the corresponding outcome with identical parameter set (T = 500 000 µs; α = 0.95; ρ = 0; POV: node 5). The sniffer node 5 always started at the very beginning of the simulation (i.e., at time 0 µs).

well as randomized start up orders with certain restrictions in terms of the supported time span.

For this reason, we configured our generator script from Section 2.3.4 accordingly. However, to keep the set of randomized simulation models manageable, some regulation regarding the time span is required. In [38], Choochaisri suggests that all nodes should start up within the first period, i.e., within $[0, T]$. Depending on the size of a node's neighborhood as well as the period length, this constraint may be too restrictive. Therefore, we want all nodes to start up within a small number of periods. Then, the real start up time for each node is randomized within this time span using the Perl instructions from Listing 7.2. Notably, the Pseudo Random Number Generator of Perl is initiated (cf. Line 2) by the same (default) seed (cf. Line 1) as the PRNG used in our simulator extDeSIMc from Section 2.3. With it, this approach allows repeatable results.

However, to be comparable to the results from the related work and from Section 4.3.2 in particular, we harmonized our creation process to comply with the proposal of Choochaisri in [38], where all nodes have to start up within the first period. This is the reason why we did use `$startupPeriods = 1` in Line 3 for the simulation models with randomized start up times within this section.

```
1 $seed = 8690401185424030;  # init seed
2 srand($seed);  # init RNG
3 $startupTime = int( rand($startupPeriods * T) );  # set start up time
```

Listing 7.2: Perl instructions to generate randomized start up times within a fixed number (`$startupPeriods`) of periods.

7.2.4 Results

As mentioned in the previous section, we simulated the possible swapped starting times as listed in Table 7.2 with the start up order $(4, 1, 2, 3)$ as initial value. For the purpose of comparison, we also simulated the start up order $(1, 2, 3, 4)$. Concluding, we could not observe further convergence behavior in addition to that one mentioned in Sections 7.1 and 7.2: The

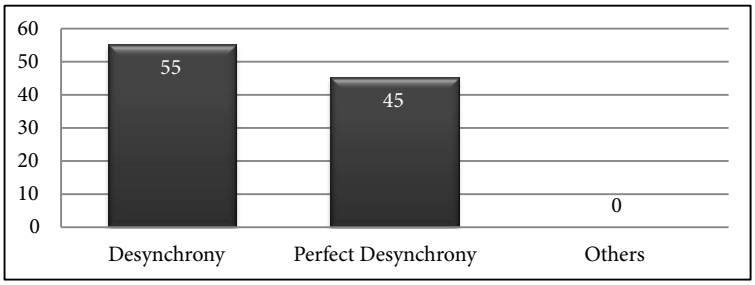


Figure 7.5: Result after 100 simulation runs of topology A_{4S} with randomized start up times.

system always reached the stable state of (perfect) desynchrony according to the dimensioning rules R1 and R2 from Section 7.1.4. At no time, the system ended in dimensioning rule R3, e.g., in fluctuation or even chaotic behavior.

However, for certain start up orders, we could observe an additional outcome, which looked different to those exemplified in Figure 7.2(b) and in Figure 7.4(a), respectively. This novel behavior is shown in Figure 7.4(b): The whole system rotates at constant speed along the period T (quite similar to Figure 7.2(b)). Indeed, in this case the whole system rotates backwards from the sniffer's perspective. Still, the temporal distance between each pair of nodes remains constant. According to dimensioning rule R2, the system reached the stable state of desynchrony.

To support this theory, we arranged further simulation runs. In particular, we this used the same topology A_{4S} as described in Section 7.1.4 but randomized start up times and start up order, respectively. Using the randomization procedure as described in Section 7.2.3, i.e., the nodes start up at randomized times within the first period, we simulated 100 different simulation models based on topology A_{4S}. The outcome of these simulations is depicted in Figure 7.5: As a result, 55 simulation runs reached the stable state of desynchrony, whereas 45 simulation runs even perfectly desynchronized. Moreover, no simulation model diverged or resulted in chaotic behavior. In fact, we could not observe any further convergence behavior in addition. To sum up, the system always reached the stable state of (perfect) desynchrony. This result argues for the applicability of our protocol – at least for this particular topology, but with randomized start up order as well as start up times.

As expected, the restriction on the time span to select start up times by random was quite useful: First, especially the automated creation of a simulation model using the Perl script from Section 2.3.4 was simplified a lot. Next, the outcome of these different simulation runs remains comparable due to this limitation. Finally, using the same seed for the Pseudo Random Number Generator, the simulation models can be re-enacted. Therefore, to check certain thresholds or to test the limits of a (protocol) parameter by simulation, we will use a randomized start up order within a fixed number of periods. Next, we will analyze the protocol parameters.

7.3 Jump Size Parameter

The basic algorithm of the midpoint approach was introduced in Section 4.3. This algorithm is quite simple in computation and easy to configure: Since the period T is affected by the size of the network (cf. Lemma 4.6 and Lemma 4.13, respectively), the jump size parameter $\alpha \in [0,1]$ (cf. Eq. (4.14b)) is the only adjustable protocol parameter for the midpoint approach and, consequently, the EXTENDED-DESYNC protocol (cf. Item F1 in Section 7.2). Nevertheless, this damping factor has a high impact on the convergence rate of the system as it controls the exponentially weighted moving average (cf. Observation 4.11). Therefore, the jump size parameter should be set to its optimum.

7.3.1 Object of Investigation

As already mentioned, the jump size parameter is a very important parameter for the midpoint approach – and thus for the derived communication protocols. Hence, Degesys et al. try to find in [50] an optimum value for this damping factor α. Due to the iterative procedure of the algorithm, there is no analytical solution available at present. Thus, the authors in [50] try to empirically identify an optimum value for the jump size parameter α – at least for single-hop topologies. As a result, they suggest to use a value close to 1 for a fast convergence of the system in case of single-hop topologies. Indeed, some publications use $\alpha = 0.9$, e.g., [49], others use $\alpha = 0.95$, e.g., [141, 50], for their experiments and in their simulations.

In addition, there is – to the best of our knowledge – no research on an optimal value of the jump size parameter for multi-hop topologies, yet. Due to the complexity of multi-hop topologies (cf. Section 4.2.2), it even seems likely that there will be no universal value for the jump size parameter which fits any potential multi-hop topology. This means that the existence of such an optimal value for multi-hop topologies is uncertain at present.

Since there seems to be no chance to find an analytical solution for the optimum value of the jump size parameter (cf. Section 4.3.2), we want to verify the suggested optimum value of the jump size parameter for single-hop topologies by experiments first. Next, we want to simulate different multi-hop topologies with various values for the jump size parameter α. In this manner, we try to find an optimum and universally valid value or – in contrast – at least an indication for its unavailability.

7.3.2 Expectation

Based on the results of the related work on the jump size parameter for single-hop topologies, we expect to verify this optimum value by means of our real-world experiments. In particular, the outcome of our experiments should equal the values stated in the related work, i.e., $\alpha = 0.9$ or $\alpha = 0.95$.

Moreover, using the experimental setup, we also want to confirm the approach to alleviate the stale information problem for single-hop topologies from Section 6.2.1. Therefore, when each node estimates its next time of firing right after the reception of its successor's firing, we expect a positive effect in comparison to the estimation immediately after the node's own firing. Nevertheless, using the stale information of the latter approach we expect the system needs longer to reach the stable state.

Again, for multi-hop topologies there is no comprehensive data available. Therefore, our expectation regarding the optimum value for the jump size parameter for multi-hop topologies is not fixed: On the one hand, the outcome of our multi-hop simulations could indicate that there is no optimum value which is universally valid. This could be expressed by different convergence rates on different topologies. This result seems likely due to the complexity of multi-hop topologies (cf. Section 4.2). On the other hand, the results of our simulations could even provide an indication of an optimum value for the jump size parameter – at least for the simulated topologies and scenarios. This could be expressed by the same optimum convergence rate based on an identical value for the jump size parameter valid for all (simulated) multi-hop topologies.

Indeed, per se each node does not have any (a priori) knowledge about its neighborhood, its constraint graph, and eventually the network topology (at start up). Therefore, a particular initial value for the jump size parameter α anyhow is required for the implementation of the midpoint approach. Hence, there has to be a default value which is independent of the underlying topology. In the end, we expect to get (at least an indication for) such a default value for the jump size parameter α.

7.3.3 Procedure

To verify the optimum value for the jump size parameter for single-hop topologies, we adopt the results from the real-world testbed primarily deployed and described in detail by Keupp in [95][6]: The testbed C_{4S} consists of five SNoW[5] sensor nodes, in particular four fully-connected nodes and one sniffer for logging purpose as depicted in Figure 7.6. To get roughly repeatable results, all nodes are powered by a switchable multi-outlet power strip. Powering on all nodes of the real-world testbed ensures an identical base line for all the distributed and local timelines of the nodes. In this regard, please recall also our recommendations in Section 5.8. As suggested by Choochaisri in [38], all nodes start up within the first period. As described in Section 5.8.1, the particular starting time depends on the node's identifier – except for the sniffer, which always starts up immediately.

The procedure of the tests is as follows:

1. The initial value for the period is set to $T = 500\,000\,\mu s$.
2. The initial value for the jump size parameter is set to $\alpha = 0.1$.
3. The nodes perform the EXTENDED-DESYNC protocol. Consequently, there is no refractory threshold, i.e., $\rho = 0$.
4. Every 100 periods[7], each node of the complete subgraph C_4
   - drops its current list of neighbors,
   - increases α by 0.1 (up to $\alpha = 1$),
   - resets its start up time, and

[6] Bachelor thesis conducted in conjunction with this work.
[7] We assume that this amount of periods is sufficient to reach the stable state of (perfect) desynchrony – if at all.

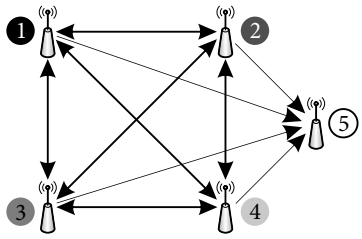


Figure 7.6: The topology C_{4S} consists of the set $N = \{1, \ldots, 5\}$ of nodes. Notably, node 1 to node 4 form the complete subgraph C_4, node 5 (sniffer) is just receiving.

- pauses any transmission for 5 periods[8] to restart the desynchronization procedure.

In addition, to analyze the impact of the stale information problem for single-hop topologies as mentioned in Section 6.2.1, we rerun this experiment with slight modifications. Now, each node estimates its next time of firing not after the reception of its successor as stated in Section 6.2.1, but immediately after its current firing. This "early" estimation will not be adjusted – even after the reception of succeeding firings. Thus, the last known time of firing of the node's successor from the previous period will be considered (cf. Eq. (4.14)). Indeed, this information is "stale" (cf. Figure 6.1 on Page 111). To sum up, stale information about the node's predecessor as well as stale information about the node's successor is used for this estimation of the node's next time of firing.

For multi-hop topologies, there is neither a guidance value nor a comparative value for the jump size parameter α available. Therefore, we transfer the procedure from above used for single-hop scenarios for multi-hop topologies. In particular, we simulated different multi-hop topologies with the identical parameter set from above, i.e., period $T = 500\,000\,\mu s$, refractory threshold $\rho = 0$, and the initial value for jump size parameter $\alpha = 0.1$. Indeed, the values for the jump size parameter α increase similar to the single-hop scenarios. In addition, for the multi-hop simulations we assume idealized conditions, i.e., all communication links are bidirectional and reliable, not any node will fail, and there is no clock drift. The start up time for each node is randomized within the first period since system start. In particular, we analyzed a star graph S_4 (cf. Definition 2.20) consisting of four sensor nodes[9], a circle graph R_{4S} (cf. Definition 2.22) consisting of four sensor nodes plus one sniffer, a line graph L_{4S} (cf. Definition 2.21) of four sensor nodes plus one sniffer, and the dumbbell graph D_7 from Section 6.2.2 consisting of seven sensor nodes (without sniffer) as depicted in Figure 6.3.

7.3.4 Results

As expected, all single-hop experiments reached the stable state of desynchrony quite fast. The system behavior for values, which are close to the boundaries of the interval of the jump size parameter, was insufficient: For instance, setting $\alpha = 0.1$ results in a slow convergence

[8] We assume that this amount of periods is sufficient to detect the restart of the desynchronization procedure easily but definitely.

[9] Since the there is no need for an additional sniffer.

rate, whereas setting $\alpha = 1.0$ results in a settling phase with lots of fluctuations. In Figure 7.7(a), an excerpt illustrates the system behavior for the three values $\alpha = 0.8$, $\alpha = 0.9$, and $\alpha = 1.0$ of our single-hop real-world testbed, when each node adjusts its next time of firing just after the reception of its successor's firing packet. For this particular topology C_4, there are no more than just marginal deviations between the outcome when the jump size parameter is set as $\alpha = 0.8$ and as $\alpha = 0.9$. Notably, the system desynchronizes even though the jump size parameter was set to its upper bound $\alpha = 1.0$. However, it looks as if setting the jump size parameter $\alpha = 0.9$ may be the optimum value for single-hop topologies. In contrast, Figure 7.7(b) presents the outcome of our real-world testbed, when a node is calculating its next time of firing immediately after its own firing. This excerpt of our single-hop real-world testbed illustrates the system behavior for the three values $\alpha = 0.8$, $\alpha = 0.9$, and $\alpha = 1.0$. For this particular topology C_4, the deviations between the different settings for α are obvious. Seemingly, setting $\alpha = 0.8$ looks best here.

Consequently, we definitely can confirm the approach from Section 6.2.1. In particular, node $i \in N$ has to calculate its next time of firing t_i^+ not immediately after the transmission of its own firing packet at time t_i (cf. Figure 7.7(b)), but instead immediately after the reception of the first subsequent firing packet of its successor $s(i)$ at time $t_{s(i)}$ (cf. Figure 7.7(a)).

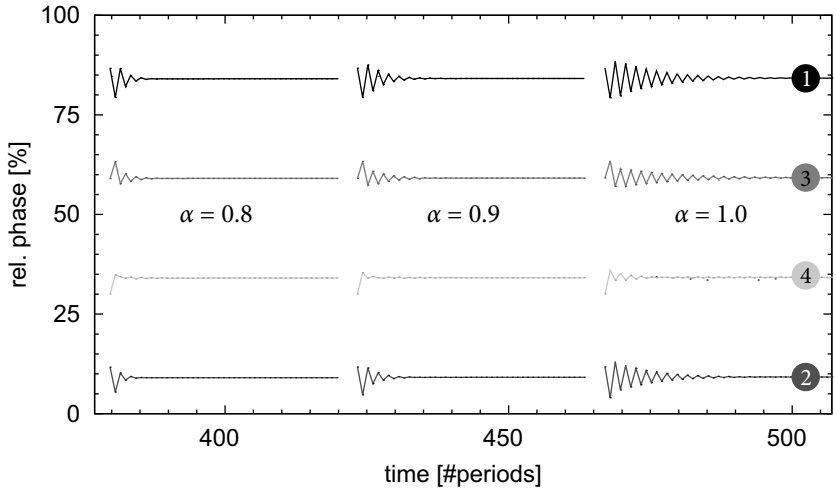
Despite the dumbbell topology D_7, the same applies to the outcome of our simulations of the analyzed multi-hop topologies, namely star graph S_4, circle graph R_{4S}, and line graph L_{4S}. All these multi-hop topologies showed quite a similar behavior and a similar convergence rate for the corresponding values of α. In particular, there are also just marginal deviations in comparison to the outcome of the single-hop experiments.

When a node starts up, usually it has no knowledge about its network and its topology, respectively. Therefore, the initial value of the jump size parameter has to suit for both, single-hop as well as multi-hop topologies. Even though the analyzed scenarios here neither cover all possible constellations of sensor nodes nor include all potential topologies, the outcome of this analysis demonstrates a sense of direction for the initial value of the jump size parameter α. Hence, and as stated in some related work (e.g., in [141, 50]), we also recommend to set $\alpha = 0.95$ as initial value of the jump size parameter for single-hop as well as multi-hop topologies.

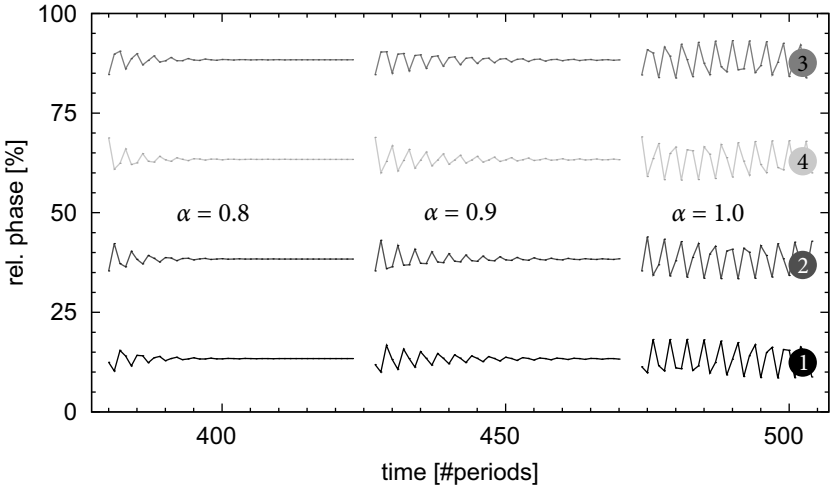
However, while analyzing the jump size parameter for multi-hop topologies, the dumbbell graph D_7 sometimes did not desynchronize. Due to the stale information problem (cf. Section 6.2.2), the system of the dumbbell topology rather diverges than converges. In fact, the time of transmission of each node fluctuates with a constant but individual amplitude, likewise shown in Figure 6.4. This fluctuating behavior is irrespective of the particular value of the jump size parameter. Hence, we will analyze the refractory threshold in the next section to possibly solve the stale information problem.

7.4 Refractory Threshold

One drawback of the midpoint approach is the stale information problem. This problem was first described and analyzed for single-hop topologies by Degesys et al. in [50]. Unfortunately, this problem is inherent to the primitive of desynchronization (cf. Section 6.2). As already indicated in Section 6.2.2, the impact of this problem is intensified in multi-hop topologies.



(a) The next time of firing is calculated after the reception of the firing packet of the node's successor. In particular, the outcome for $\alpha = 0.8$, $\alpha = 0.9$, and $\alpha = 1.0$ is shown.



(b) The next time of firing is calculated immediately after the node's own firing. In particular, the outcome for $\alpha = 0.8$, $\alpha = 0.9$, and $\alpha = 1.0$ is shown.

Figure 7.7: Excerpts of the outcome of our real-world testbed analyzing the jump size parameter for topology C_{S4}, POV: node 5 (sniffer).

As a consequence, also the EXTENDED-DESYNC protocol suffers from stale information (cf. Section 6.5).

To solve this issue, we have introduced the *refractory threshold* in Section 6.3. In Section 6.3.2, we made a first and rough estimation of the impact of this probabilistic parameter;

a deeper analysis is still missing. Since the value of the refractory threshold is set once during initialization (likewise the jump size parameter α), a reasonable initial value has to be identified. However, such a (default) initial value has not been specified so far. Therefore, we also try to discover an optimal value for this probabilistic parameter ρ (for multi-hop topologies) by means of simulations.

7.4.1 Object of Investigation

The simple dumbbell topology D_7 in Section 6.2.2 impressively demonstrates the implication of the stale information problem in multi-hop topologies. For this reason, we introduce a probabilistic parameter in Chapter 6. In particular, we did add in Section 6.3 the refractory threshold ρ to the EXTENDED-DESYNC protocol to eventually improve it as EXTENDED-DESYNC$^+$ protocol.

Indeed, for the refractory threshold there is no optimal (initial) value available at present. Due to the complexity of multi-hop topologies and similar to the damping factor α, there may be no analytical solution for an optimal value of this probabilistic parameter ρ (cf. Section 7.3.1). Therefore, it even seems likely that there will be no universal value for the refractory threshold which fits any potential multi-hop topology. In contrast to the jump size parameter, there do exist neither reference values nor comparable results from former analyses. Hence, we have to try to find an (seemingly) optimal initial value from scratch by means of simulations or experiments.

7.4.2 Expectation

The main challenge to find an optimum (initial) value for the refractory threshold ρ is the absence of reference values. Indeed, Eq. (6.1) in combination with the domain of the refractory threshold ρ, and especially its boundaries give some guidance: As already mentioned in Section 6.3.1, setting $\rho = 0$ lets a node always adjust its next time of firing according to its adjustment function. In contrast, setting $\rho = 1$ forces a node to not use the adjustment function anymore. Hence, there will be a trade-off between fluctuating behavior and a prolonged convergence phase.

Consequently, for values of the refractory threshold ρ close to 0, we expect the system to result in the same convergence behavior like using the EXTENDED-DESYNC protocol (without this probabilistic parameter). This may be even a fluctuating and rather chaotic behavior (cf. Section 6.2.2). Whereas, for values of the refractory threshold ρ close to 1, we expect the system to converge after a quite long convergence phase – if at all. Thus, the closer the value of the refractory threshold is to 1, the longer takes the convergence phase of the system. Finally, recalling the short impact analysis from Section 6.3.2, we expect the system to desynchronize for certain values of the refractory threshold – at least for $\rho = 0.25$ as indicated in the sample scenario in Section 6.3.2.

7.4.3 Procedure

For a deeper analysis of the refractory threshold ρ, we reconsider the sample scenario as introduced in Section 6.2.2: This scenario is based on a dumbbell topology D_7 as reillustrated in Figure 7.8. In particular, the network consists of two complete (as well as cyclic) subgraphs,

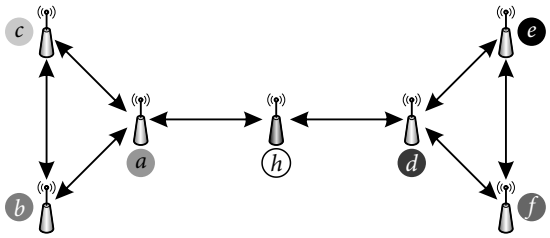


Figure 7.8: The reillustrated topology D_7 consists of the set $N = \{a, \ldots, f, h\}$ of nodes with bidirectional links.

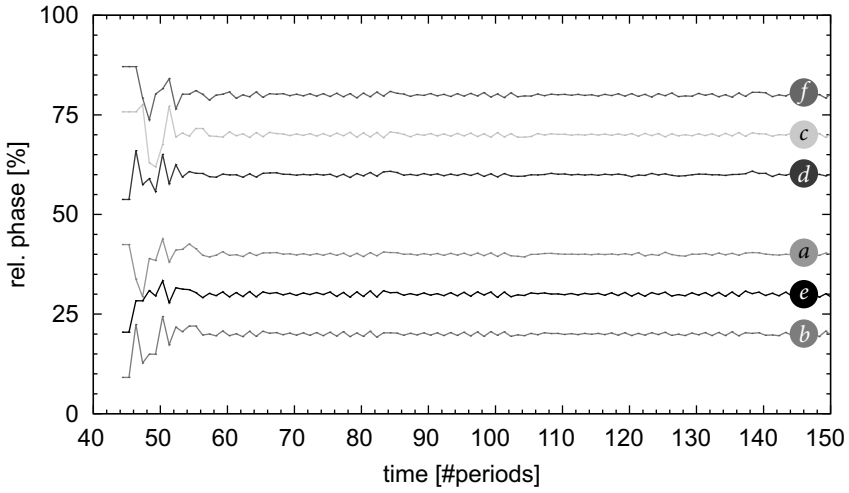
which are connected through node h. Consequently, without node h there remain two disjoint connected components, and thus two independent networks. Please note that we do not install a sniffer in addition, since $N_C(h) = N$ holds. Hence, we primarily select node h for monitoring the system behavior.

For an efficient and fast analysis, we created and executed the simulation models as follows: All simulations are based on the same parameter set to get comparable results. In particular, we set period $T = 1\,000\,000\,\mu s$, jump size parameter $\alpha = 0.95$, and the initial value for the refractory threshold $\rho = 0.0$. As suggested by Choochaisri in [38] and as a result from Section 7.2, the nodes of these two subgraphs start up within the first period – except for node h: To better examine the impact of the refractory threshold, we utilized node h as "connecter", i.e., this node starts up just a few periods after the (perfect) desynchronization of each of both subgraphs. This is an important prerequisite to enforce the fluctuating behavior from Section 6.2.2, and thus, to focus on the analysis of the stale information problem.

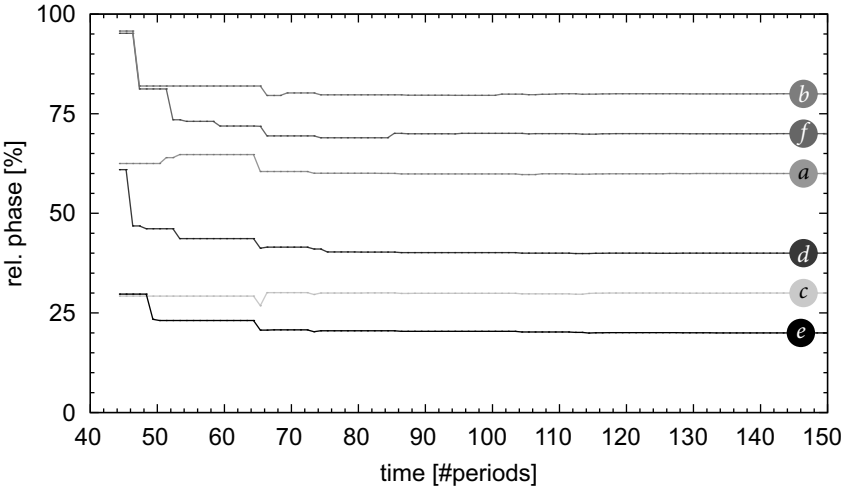
Since the refractory threshold ρ may be stated as parameter `<phaseKeeping>` within the `<Simulation>` element of a simulation model (cf. Section 2.3.3), we may define different simulation elements for the same topology within one file. To obtain comparable simulation results, we reuse the list of events within the different `<Simulation>` elements of one simulation model. Hence, we only have to adjust the refractory threshold ρ as attribute of the `<phaseKeeping>` element within each `<Simulation>` section of a simulation model. This means that we will use different values for the refractory threshold and keep the (randomized) multi-hop topology as well as the rest of a simulation configuration unchanged.

7.4.4 Results

As expected, the nodes of the network rather fluctuate than desynchronize when setting the refractory threshold $\rho = 0$. In fact, the outcome was similar or even identical to the outcome shown in Figure 6.4 on Page 114. However, slightly increasing the value of the refractory threshold results in a decreasing amplitude of the fluctuation. Nevertheless, the system may still not end up in the stable state of (perfect) desynchrony. Indeed, the nodes of the two subgraphs in topology D_7 still behave like the teeth of two gear wheels moving back and forth all the time. For instance, Figure 7.9(a) exemplifies the outcome of one simulation run of topology D_7 with refractory threshold $\rho = 0.1$ from the POV of node 7 since its start up in period 45. Please note that the rest of topology D_7, i.e., the two subgraphs, are desynchronized at



(a) Excerpt of the outcome of a simulation run of topology D_7 (T = 1 000 000 µs; α = 0.95; ρ = 0.1; POV: node h).



(b) Excerpt of the outcome of a simulation run of topology D_7 (T = 1 000 000 µs; α = 0.95; ρ = 0.9; POV: node h).

Figure 7.9: Excerpts of the undesired outcome of a simulation run of topology D_7 analyzing the refractory threshold, POV: node h.

this particular period. This is indicated in Figure 7.9 by the short horizontal lines on the left side of the corresponding line chart. These horizontal lines are the direct result of node h's listening time before its first own firing as introduced in Section 5.8.1.

Also as expected, the convergence is prolonged for high values. For instance, setting ρ =

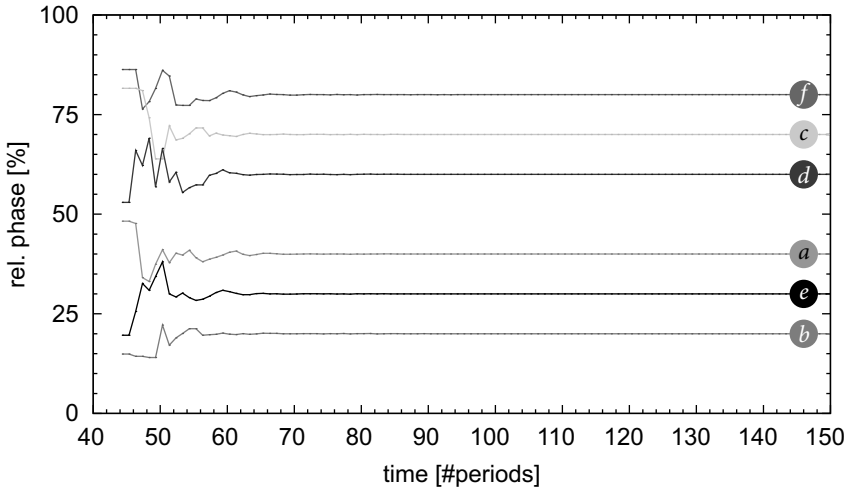


Figure 7.10: Excerpt of the outcome of a simulation run of topology D_7 analyzing the refractory threshold (T = 1 000 000 µs; α = 0.95; ρ = 0.25; POV: node h).

0.9 means that node $i \in N$ will adjust its next time of firing t_i^+ only in one out of ten periods. Figure 7.9(b) exemplifies one outcome of the simulation run of topology D_7 with refractory threshold ρ = 0.9 from the POV of node 7 since its start up in period 45. Here, the system reaches the stable state of perfect desynchrony eventually after about 60 periods since the joining of node h. Nevertheless, the system utilizing the refractory threshold is not inferior in quality to the system without using this probabilistic parameter.

As already mentioned in Section 6.3.2, the system desynchronizes when setting ρ = 0.25. Thus, the probabilistic parameter refractory threshold definitely may help nodes in multi-hop topologies to solve the stale information problem and eventually to escape the "vicious circle" of fluctuating (and probably chaotic) behavior. So far, the system eventually reaches the stable state of (perfect) desynchrony due to a serviceable protocol configuration. However, another interesting criterion is the convergence rate. In Figure 7.9(b), the system reaches the stable state of perfect desynchrony after about 60 periods since node h joined the network. To interpret the performance of the EXTENDED-DESYNC$^+$ protocol, we compare this outcome of a simulation utilizing ρ = 0.9 to that one using ρ = 0.25. Thus, Figure 7.10 exemplifies the convergence behavior of the system based on the same configuration as used for Figure 7.9, despite the refractory threshold is set as ρ = 0.25: Although each node skips every forth calculation of its next time of firing, the system reaches the stable state of perfect desynchrony after about 20 periods since node h joined the network. This is much faster and suits our expectations much better.

Likewise for the jump size parameter, the initial value of the refractory threshold has to suit for both, single-hop as well as multi-hop topologies. Even though, the analyzed scenarios neither cover all possible constellations of sensor nodes nor include all potential topologies, the outcome of this analysis demonstrates a sense of direction for the initial value of the refractory threshold ρ. As demonstrated by the simulation of topology D_7, setting the re-

fractory threshold ρ too close to 0 may be not sufficient (cf. Figure 7.9(a)). Since the system seems not to benefit from higher values for the refractory threshold, setting $\rho = 0.25$ as initial value of the refractory threshold looks like a good candidate. Therefore, we recommend to use $\rho = 0.25$ as initial value for the refractory threshold of the EXTENDED-DESYNC$^+$ protocol. Nevertheless, this result is mainly based on the analysis of the particular dumbbell topology D_7 from Figure 7.8. In the next section, the EXTENDED-DESYNC$^+$ protocol and the refractory threshold, respectively, will be subject to further tests against our requirements from Section 1.2.

7.5 Applicability

In Section 1.2, we stated our demands which have to be met by our protocol. The analyses within the previous section, indicate that our EXTENDED-DESYNC$^+$ protocol fulfills at least some of our requirements. Nevertheless, a concluding analysis in this regard is missing. Therefore, in this section we verify the statement from Section 1.2, i.e., our EXTENDED-DESYNC$^+$ protocol is *"a lightweight and applicable as well as scalable MAC protocol, which implements a self-organizing and dynamic TDMA schedule. This communication protocol not only has to support arbitrary (multi-hop) topologies and has to be robust against environmental perturbations, but also has to converge efficiently to an adapted schedule in case of topology changes"*.

7.5.1 Object of Investigation

As mentioned before, we verified in Sections 7.1 to 7.4 some of our requirements from Section 1.2. In particular, our EXTENDED-DESYNC$^+$ protocol is *lightweight*, as it requires just low computational effort and a small overhead due to

- the implementation of the simple midpoint approach from Section 4.3 as well as
- our phase shift propagation approach from Section 5.3 to solve the hidden terminal problem. In addition, it
- neither relies on expensive lookup tables
- nor does its realization contain complex calculations.

Moreover, the numerous real-world testbeds (cf. Sections 7.1 to 7.4 as well as, e.g., [124, 125, 126]) which are built upon real sensor nodes from Section 2.4, e.g., SNoW^5, prove the *applicability* of our protocol for Wireless Sensor Networks. This means that even sensor nodes, which are in general severely limited in computational power as well as in memory (cf. Definition 2.6), are able to execute our EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocols. Furthermore, our communication protocols do not rely on a fixed infrastructure nor on a central control unit[10]. Instead, the midpoint approach and the phase shift propagation approach are realized just by means of locally available information. Thus, the EXTENDED-DESYNC$^+$ protocol implements a *self-organizing* and dynamic *TDMA schedule*.

[10] As a sniffer just receives but does not transmit packets itself, we do not consider a sniffer as central control unit.

As a result, we will have to demonstrate within this section the remaining requirements from Section 1.2. In particular, we will show that our self-organizing protocol is *scalable*, supports *arbitrary (multi-hop) topologies*, and is *robust* against environmental perturbations. In case of topology dynamics our EXTENDED-DESYNC$^+$ protocol *converges efficiently* to an adapted schedule. With it, our communication protocol will meet all our demands from Section 1.2.

7.5.2 Expectation

As indicated in the previous Section 7.5.1, we expect our EXTENDED-DESYNC$^+$ protocol to meet all our demands from Section 1.2. Indeed, we may expect a prolonged convergence phase for very complex multi-hop topologies – especially when there are topology dynamics. However, we do not expect any fluctuations or even collisions. In contrast, we expect any analyzed system to eventually reach the stable state of (perfect) desynchrony – at least in phases without environmental perturbations but regardless of the underlying topology.

7.5.3 Procedure

It is impossible to simulate all potential (multi-hop) networks. In addition, it is also impossible to simulate all possible environmental perturbations. Nevertheless, to cope with these two issues, we will focus on randomized graphs to cover a large variety of different multi-hop topologies. In this regard, we make intensive usage of our generator script from Section 2.3.4: Depending on the network size, the randomly generated network typically results in a multi-hop topology. To get comparable results, we mainly set the initial values of the protocol parameters jump size parameter $\alpha = 0.95$ and refractory threshold $\rho = 0.25$.

To receive an impression about the relation of the number of nodes and the number of simulation scenarios to the size of the simulation model, Table 7.3 lists the file size of selected simulation models (cf. Section 2.3.3). With respect to the hardware limitations of the simulating PC, simulation models of a network with up to 100 nodes are not only well manageable but also sufficient and feasible to demonstrate the system behavior. Therefore, we set up randomized multi-hop topologies consisting of up to 100 nodes and up to 100 simulation scenarios maximum. Moreover, and according to Section 7.2.4, the start up times of the nodes are set randomly within the first few periods.

To analyze the robustness of the EXTENDED-DESYNC$^+$ protocol against environmental perturbations, we manually modify the event list of at least one node in some of the randomized simulation scenarios by means of `ON`, `OFF`, and `DEAD` events (cf. Section 2.3.2): The `OFF` event simulates an arbitrary and temporary power down of a node, e.g., due to low battery. To reactivate such a "sleeping" node later during the simulation run, we use the `ON` event. To definitely remove a node from the network (and thus from the simulation), we utilize the `DEAD` event. The usage of these events may even be used to emulate mobility of the nodes to some extend (cf. Section 8.1.3).

7.5.4 Results

As mentioned in Section 7.5.3, we extensively used our generator script from Section 2.3.4 to generate these multi-hop simulation models. In this regard, we simulated networks con-

#Nodes	#Simulations	File Size
10	10	40.3 kB
10	100	293.6 kB
100	10	417.2 kB
100	100	1.9 MB
1000	10	4.0 MB
1000	100	19.0 MB
5000	10	20.0 MB
5000	100	95.5 MB

Table 7.3: File size of randomized simulation models in relation to the number of nodes and the number of simulation scenarios.

sisting of just five nodes (e.g., topology A_{4S} from Section 7.1.4) up to networks of 100 nodes (e.g., within this section). According to Section 7.2.4, all nodes of these randomized simulation scenarios start up randomly within the first few periods. As expected, some of these network scenarios reached the stable state of (perfect) desynchrony, although the refractory threshold was set to $\rho = 0$. However, setting jump size parameter $\alpha = 0.95$ and refractory threshold $\rho = 0.25$, all these randomized networks being subject of investigation eventually reached the stable state of (perfect) desynchrony – at least when there are no topology dynamics. Consequently, due to the randomized generation procedure, we may state that our EXTENDED-DESYNC$^+$ protocol supports arbitrary (multi-hop) topologies. Moreover, due to the different network sizes, we may also state that the EXTENDED-DESYNC$^+$ protocol is scalable.

In addition and as described in the previous section, we also emulate environmental perturbations. Indeed, immediately after such a perturbation an already desynchronized system may not be desynchronized for some periods. However, we detected no collision or packet loss, neither when a node joined the network nor when a node left the network. Nevertheless, after a short settling phase, the simulated networks again reached the stable state of (perfect) desynchrony – at least for jump size parameter $\alpha = 0.95$ and refractory threshold $\rho = 0.25$. Hence, due to the re-attainment of the stable state of (perfect) desynchrony, we may further state that the EXTENDED-DESYNC$^+$ protocol is robust against topology dynamics.

To support our statements from above, we present in detail the behavior of a sample scenario (cf. [125]): Especially, we discuss (excerpts of) the outcome of the randomly generated topology A_{100} consisting of the set $N = \{1, \ldots, 100\}$ of nodes without a sniffer. Here, the links between the nodes are symmetric and set randomly. All nodes start up within the first three periods. The protocol parameters of this scenario are: period $T = 10\,000\,000\,\mu s$, jump size parameter $\alpha = 0.95$, and refractory threshold $\rho = 0.25$. To illustrate the system behavior on topology perturbations and, notably, leaving and re-joining nodes, we focus on the behavior of one specific node, namely node $82 \in A_{100}$. The constraint graph $N_C(82)$ of node 82 is shown in Figure 7.11. First, we switch off (i.e., `OFF` event) one three-hop neighbor (i.e., node 25), next one two-hop neighbor (i.e., node 9), and finally one one-hop neighbor (i.e., node 15) of node 82. These three nodes will be switched on (i.e., `ON` event) and thus re-join the network after 45 periods of downtime:

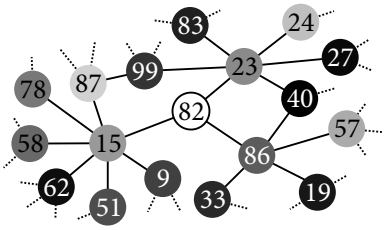


Figure 7.11: The constraint graph $N_C(82)$ of node 82 within topology A_{100} consisting of the set $N = \{1, \ldots, 100\}$ of nodes.

- The three-hop neighbor node 25 leaves at period 15 and rejoins at period 60,
- the two-hop neighbor node 9 leaves at period 30 and rejoins at period 75, and
- the one-hop neighbor node 15 leaves at period 45 and rejoins at period 90.

The corresponding excerpt of the simulation outcome from the point of view of node 82 is illustrated in Figure 7.12: First, the three-hop neighbor node 25 is leaving the network at period 15. Since this node 25 is not part of the constraint graph $N_C(82)$ of node 82 (cf. Figure 7.11), the leave of this node has no measurable impact on node 82 (as expected). Next, the two-hop neighbor node 9 is leaving the network at period 30. As described in Section 5.8.2, this two-hop neighbor is "virtually" kept by node 82 for a few periods. Afterwards, this two-hop neighbor will not be considered anymore by node 82 (cf. Figure 7.12). Finally, the one-hop neighbor node 15 together with the two-hop neighbors node 87, node 78, node 62, node 58, node 51, and node 9 of node 82 (cf. Figure 7.11) are leaving the network at period 45.[11] Again, node 15 as well as all corresponding two-hop neighbors of node 82, which are propagated by this one-hop neighbor, are virtually kept for a few periods. As displayed in Figure 7.12, the leave of this one-hop neighbor node 15 has a high impact on the network and especially on node 82. In particular, this part of the the network tries to desynchronize again around period 50. Nevertheless, after a short settling phase of about 10 periods, the system is once again in the stable state of desynchrony.

As mentioned before, switching on the three-hop neighbor node 25 at period 60 has no measurable impact on the behavior of node 82. Since the connecting one-hop neighbor node 15 still is powered off, switching on the two-hop neighbor node 9 at period 75 has also no measurable influence. Finally, when the one-hop neighbor node 15 re-joins the network, the corresponding two-hop neighbors are also visible for node 82. As expected, the re-entry of this one-hop neighbor has a high impact on the behavior of the system and of node 82 in particular (cf. Figure 7.12). However, the system reaches the stable state of desynchrony about 20 periods after the re-joining of node 15 at period 110. In addition, we detect no collision or packet loss, neither when a node joined the network nor when a node left the network.

[11] Please note that node 9 has already left the network at period 30

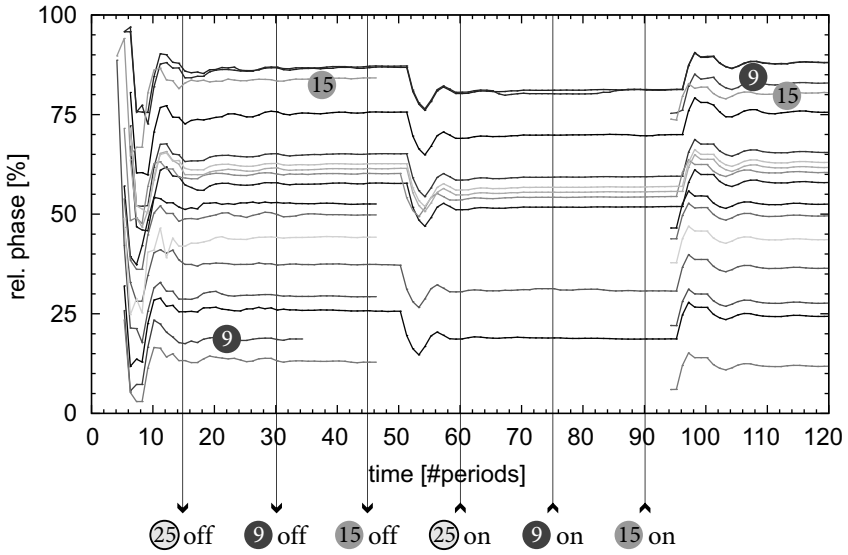


Figure 7.12: Excerpt of the outcome of the simulation of topology A_{100} (T = 10 000 000 µs; α = 0.95; ρ = 0.25; POV: node 82).

7.6 Summary

In this chapter, we evaluated our self-organizing MAC protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ in real-world testbeds as well as simulations. Since any conclusion drawn from simulation runs is at most as reliable as the underlying simulation model permits, we had to validate our simulation models first. This validation process is discussed in Section 7.1. Here, we oppose the results from a specific and well-configured real-world testbed to the results of the corresponding simulation model. This iterative approach not only led us to a valid simulation model but also pointed us to another "configuration pitfall": The setup of testbed and simulation in terms of start up order may influence the simulation results.

Therefore, we next analyzed the setup difficulties in Section 7.2. In particular, we discussed the impact of different start up times as well as different start up orders. As a result, we came up with an appropriate specification on how to get comparable results. Moreover, the restriction on a certain time span for start up times simplifies the process of automated simulation model creation.

Furthermore, we analyzed the protocol parameters jump size parameter α in Section 7.3 and refractory threshold ρ in Section 7.4. First, we elaborated the initial value of the jump size parameter α = 0.95 for multi-hop topologies. This initial value is also suggested for single-hop topologies in the related work. In addition, we analyzed the new parameter refractory threshold. In this regard, we tried to find an appropriate initial value for the refractory threshold, which is feasible and suitable for single-hop as well as multi-hop topologies. As a result, we recommend to set the refractory threshold ρ = 0.25, i.e., each node should skip every forth estimation of its next time of firing.

Finally, we checked the applicability of our EXTENDED-DESYNC$^+$ protocol. Therefore, we run several simulations of our communication protocol on randomized multi-hop topologies setting jump size parameter $\alpha = 0.95$ and refractory threshold $\rho = 0.25$. In particular, we were able to show that our demands from Section 1.2 are all met – at least for the analyzed simulation models. To sum up, our EXTENDED-DESYNC$^+$ protocols is scalable, supports arbitrary (multi-hop) topologies, and is robust against environmental perturbations.

Chapter 8

Discussion

Abstract

In the last chapter, we have analyzed our self-organizing MAC protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, respectively. This analysis mainly checks for the protocol's compliance with our requirements from Section 1.2. As a result, our communication protocol showed to meet our demands – at least for the analyzed test cases and scenarios. Nevertheless, there is room for improvement. Therefore, we give a short outlook to potential enhancements regarding the protocol in Section 8.1. Moreover, our self-organizing protocol seemingly is predestined to inherently support additional (network) services. We will suggest some of these add-ons in Section 8.2.

8.1 Outlook

In Chapters 5 and 6, we have specified our self-organizing protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, respectively. The specification mainly utilizes the two protocol parameters jump size parameter α and refractory threshold ρ for configuration and tuning. Nevertheless, both parameters are intended to be fixed from system start. This means that the initial value neither will be changed nor is able to reflect specific network conditions like the changeable size of a node's constraint graph. Therefore, we will briefly discuss adaptive versions of the jump size parameter in Section 8.1.1 and of the refractory threshold in Section 8.1.2. Section 8.1.3 outlines further potential objects of investigation of our EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocols.

8.1.1 Adaptive Jump Size Parameter

The key parameter of the midpoint approach, and thus of our EXTENDED-DESYNC protocol, is the jump size parameter α. In accordance with the related work, we consider this parameter to be fixed from system start. Therefore, it is important to carefully select the best possible value at design time. Consequently, the ideal value was identified empirically for single-hop topologies by Degesys et al. in [50] and for multi-hop topologies by Keupp in [95][1]. Based on this work, we further analyzed this particular parameter in Section 7.3 for multi-hop topologies. In this regard, we identified $\alpha = 0.95$ as an ideal value for single-hop as well as multi-hop topologies.

Nevertheless, this parameter is set once and unchangeably at system start within the initialization phase. Consequently, it cannot dynamically reflect topology dynamics or any other

[1] Bachelor thesis conducted in conjunction with this work.

effect. Moreover, the jump size parameter is identical for each node of the network – irrespective of node-specific issues, e.g., the characteristic of its constraint graph.

Therefore, we first propose to introduce a node-specific jump size parameter, i.e., to substitute the individual parameter α_i for each node $i \in N$ for the common parameter α. This allows for the consideration of node-specific issues, e.g., the node's degree or the size of its constraint graph. Consequently, each node i may be equipped with an individual initial value for its jump size parameter α_i.

Next, we propose to make the node-specific jump size parameter α_i adaptable during runtime. For instance, assuming node i is in the stable state of perfect desynchrony. According to Definition 4.3, this means that the adjustment function of node i $\varphi_i\left(N_R(i), t^{++}\right) = 0$ for some periods. As we utilize the midpoint approach, according to Eqs. (4.13) and (4.14) the calculation of the next time of firing t_i^+ of node i is

$$\varphi_i\left(N_R(i), t_i\right) = 0 = \alpha_i \cdot \varepsilon_{t_i}.$$

Hence, it may be possible to set $\alpha_i = 0$ in case node i is in perfect desynchrony, and thus to simplify the calculation of node i's next time of firing.

Moreover, an adjustable α_i enables each node i to react in a more flexible way, e.g., in case of topology dynamics: Continuing the example from above, the constraint graph of node i will change and, as a result, its phase neighbor(s) as well. Consequently, this node needs to adapt its next time of firing. This means that node i has to set the value of its jump size parameter appropriately.

As discussed in Lemma 4.9 on Page 56, a so-called multiple phase neighbor (cf. Definition 4.4) could destabilize a multi-hop system. According to Observation 4.6, a node is not aware of its role as multiple phase neighbor in general. However, the node's degree may be an indicator to start with in this regard. Therefore, it may be reasonable to install the jump size parameter α_i of node $i \in N$ as a function of the size of the set of nodes $|N_C(i)|$ of its constraint graph (cf. Definition 2.28). Indeed, the jump size parameter of such a multiple phase neighbor, i.e., of a node with high degree, should be adapted more carefully in a conservative way.

In fact, a compensation strategy depending on the number of nodes is declined in [141]: Especially in highly dynamic networks, the (size of the) constraint graph is subject to frequent changes. Consequently, it may be difficult to estimate this size in an accurate way. Nevertheless, our protocol relies on the periodical firing of neighbor information (cf. Sections 5.5 and 5.6). Therefore, we still think that it is reasonable to implement an adaptive jump size parameter, which depends on, e.g., the (size of the) node's constraint graph. Nevertheless, the analysis of our proposals regarding the adaptive jump size parameter remains a subject of future work.

8.1.2 Adaptive Refractory Threshold

The key parameter of our EXTENDED-DESYNC$^+$ protocol is the refractory threshold ρ. Comparable to the jump size parameter, the value of the refractory threshold is not only set once at system start but also identical for each node of the network. This implementation of the refractory threshold suffers from the same shortcomings as the non-adaptive implementation of the jump size parameter (cf. Section 8.1.1): In particular, the current implementation

does not consider node specific issues. Moreover, as demonstrated in Section 7.4, it is difficult to find an ideal value for the refractory threshold. However, we identified $\rho = 0.25$ as good trade-off between convergence rate and damping fluctuations for single-hop as well as multi-hop topologies.

Therefore, we first propose to introduce a node-specific refractory threshold ρ_i for each node $i \in N$ by replacing the common parameter ρ. Next, we propose to make the refractory threshold adaptable during runtime: For instance, assuming node i is in the stable state of perfect desynchrony. This means that the adjustment function of node i $\varphi_i\left(N_R(i), t^{++}\right) = 0$ for some periods. In case node i is in perfect desynchrony, we suggest to set the refractory threshold of node i to $\rho_i = 1$ (cf. Section 6.3.1). This simplifies and shortens the calculation of node i's next time of firing.

Furthermore, we also propose to let the refractory threshold ρ_i of node $i \in N$ depend on node-specific conditions. Likewise for the jump size parameter, the size of the set of nodes $|N_C(i)|$ of its constraint graph (cf. Definition 2.28) may be such a node-specific condition. Indeed, it may be also feasible that the refractory threshold is a function of the current value of its adjustment factor ε_{t_i} (cf. Eq. (4.11)). With it, the state of convergence of a node would influence its refractory behavior using the refractory threshold. Nevertheless, the analysis of our proposals regarding the adaptive refractory threshold remains a subject of future work.

8.1.3 Additional Objects of Investigation

In Section 1.1, we motivated the objective of this work, i.e., the conceptional design and analysis of a self-organizing communication protocol for arbitrary topologies in Wireless Sensor Networks based on the primitive of desynchronization. In Chapter 7, we have checked whether our requirements from Section 1.2 are fulfilled by our protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$. Consequently, we are mainly interested in the protocol's convergence behavior. Thus, "classical" network issues, like routing behavior, data rate, network throughput, and channel utilization have not been analyzed in-depth, but will be subject of future work.

Regarding the energy-related performance of our EXTENDED-DESYNC protocol, there is an initial analysis in [121]. Indeed, this analysis is not in-depth and may need further considerations. Moreover, strictly speaking, we analyzed in Chapter 7 the behavior of our protocol just for one specific type of Wireless Network, namely Wireless Sensor Networks. Hence, further real-world testbeds and/or simulations have to be established for future work to extend the number of properly considered Wireless Network types.

In addition, mobility was also simplified within this work: We did respect changes in the topology, but we did not consider temporal aspects, e.g., for fast moving nodes. However, especially in Vehicular Ad hoc Networks velocity and communication range are two important parameters to make a statement on the network behavior. For instance, a first protocol based on the primitive of desynchronization but specific to (single-hop) VANETs is presented in [159]. Consequently, the analysis of these aspects mentioned within this section also remains subject of future work.

8.2 Add-Ons

Due to the periodic transmission scheme, our self-organizing protocols EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ are predestined to support additional (network) services. Especially such (network) services which implement a beacon-based approach: In general, a *beacon* is a small packet that is transmitted periodically and contains all necessary data to offer this particular service. Such beacons may be transmitted easily subsequent to a firing packet.

Therefore, due to the periodical transmission scheme of our protocols, it may be an easy task to offer additional (network) services. In this section we will exemplify some potential services, namely *time synchronization* in Section 8.2.1, *routing* in Section 8.2.2, and *distributed data management* in Section 8.2.3.

8.2.1 Time Synchronization

As already stated in Definition 2.6, each sensor node operates its own clock to record its local time of internal as well as of external events. Usually, this local clock is driven by a quartz crystal connected to the node's microcontroller. Unfortunately, the frequency stability of these crystal oscillators is influenced by several (environmental) factors. The influence of thermal factors is remarkable compared to the already existing frequency tolerance of such clock generators. For instance, the Vishay XT49S crystal [190], as a typical representative of low cost quartz crystals often deployed on sensor nodes, has a frequency tolerance of ±30 ppm at 25 °C. Its frequency tolerance over the entire operating temperature range (−10 °C to +70 °C) is ±50 ppm, and its aging effect in the first year is ±5 ppm. Especially, since different temperature gradients at sensor nodes are likely in such a distributed system, these distributed local clocks do not run synchronously in general. Consequently, a common notion of time within such systems is missing without further support. And in fact, *time synchronization* is an important service for several applications for Wireless Networks (cf. Definition 2.34).

For this reason, some methods to maintain a common notion of time within such distributed systems already exist: For instance, the interpretation of a (global) time radio signal enables a common notion of time with admissible precision for (some) nodes of a WSN – at least within a limited area around the sender. One example is the DCF77 long wave time signal, which is transmitted from Mainflingen (nearby Frankfurt/Main) and can be received within a radius of 2000 km in large parts of Europe. This time signal is used not only by many radio controlled clocks within Europe, but also by some WSN applications (cf. [36, 138]). However, each node which has to receive and to decode the DCF77 time signal requires a particular hardware unit, which consumes a significant amount of energy, is quite costly, and increases size and weight of the hosting sensor node.

Another example are (global) satellite navigation systems, like Galileo or GPS [146, 83, 52, 165, 196]. Such a satellite navigation system allows not only a precise outdoor localization of (mobile) objects, but also a precise time synchronization. In fact, the precise localization of the receiver is a basic prerequisite for a precise time synchronization: Only after the localization it is possible to determine the exact propagation delay (cf. Section 5.4.1). Consequently, both services work well just outdoors due to the required line-of-sight between satellites and receivers. Nevertheless, the precise time signal of a satellite navigation system was already utilized by some WSN applications in [91, 47, 32]. However, each node which has to receive

and to decode such a satellite signal also requires a particular hardware unit. Likewise the DCF77 time signal receiver unit, this hardware is also quite costly and consumes a considerable amount of energy.

Since the interpretation of both signals, the DCF77 time signal as well as the time signal of a satellite navigation system, is not applicable for most WSNs due to energy and cost constraints, the synchronization of local clocks of sensor nodes can be achieved by installing a particular time synchronization service (cf. Definition 2.34). However, due to the inherent instabilities of the assembled quartz crystals as well as the limited precision of such local clocks, the time synchronization process of the distributed clocks should be performed periodically (cf. [158]).

Indeed, there are design principles for time synchronization protocols, namely energy consumption, scalability, robustness (regarding topology dynamics), security and reliability, as well as ad hoc deployment (cf. for instance [55, 158]). According to Section 1.2 and Chapter 7, our self-organizing MAC protocols do meet these demands quite well. This enables our protocols to be utilized for an appropriate time synchronization: As described in Sections 5.5 and 5.6, our self-organizing protocols rely on the periodical exchange of accurate and precise timestamps (cf. Section 5.4). Certainly, these timestamps can be used to provide a common notion of time across the Wireless Sensor Network (cf. Definition 2.34). Moreover, some time synchronization protocols comprise an exploration phase. In general, this phase proceeds the proper time synchronization and has to be executed once in a while. Here, the periodical firing messages of our self-organizing protocol may facilitate such an exploration phase as well.

A first implementation of such a time synchronization approach, which is based on our EXTENDED-DESYNC protocol is presented in [123]. The synchronization error here is in the order of a few microseconds at a clock resolution of 1 µs. Moreover, an enhanced time synchronization mechanism, which is also based on our EXTENDED-DESYNC protocol but additionally allows the calculation of clock drifts, is described in [19].

8.2.2 Routing

Another important service in Wireless Sensor Networks is the *routing* of messages in multi-hop topologies: Assuming, specific data has to be transmitted from one particular sender to certain receiver(s). Since the sender might not be able to communicate directly with the intended receiver(s) in multi-hop topology networks, a *routing* service is needed. Such a routing service will select a specific path from sender to receiver within the network, correspondingly. Finally, the data will be routed along this path.

Indeed, data can also be transmitted to a distant receiver without a distinct routing service. For instance, one simple approach may be the *flooding* of the whole network: This means that the sender broadcasts its data to all nodes within its communication range. Each receiving neighbor in addition also broadcasts this information, and so on. Finally, each node of the network not only forwarded the data but also received the data. Consequently, the intended receiver(s) did receive the data as well. However, this approach is very costly due to the multiple (and probably recurring) transmission (i.e., reception and forwarding) of data. Therefore, sophisticated and more efficient routing protocols do exist. For an overview on routing approaches and protocols for Wireless Sensor Networks see for example [4, 56, 57, 103, 130, 166].

One principle of our self-organizing protocol is the phase shift propagation, i.e., the exchange of neighbor information (cf. Section 5.3). In addition, each node is able to create its constraint graph (cf. Section 5.3.2). Consequently, to some extent, each node also has knowledge about its two-hop neighborhood. Due to the periodic transmission of neighbor information (i.e., firing messages), this knowledge is updated continuously. Remarkably, some routing protocols are based on this information. Therefore, it should be feasible to extend our protocol to support routing protocols (at higher levels).

For instance, the *Optimized Link State Routing* (OLSR) is described in [40]. This routing protocol uses so-called *Hello* as well as *Topology Control* (TC) messages to discover and upgrade the node's one-hop as well as two-hop neighborhood information. According to this information, the OLSR service will establish a route from sender to receiver. Moreover, the *Statistic-Based Routing* (SBR) in [97] is also based on periodically transmitted *Hello* messages. However, this protocol implements a statistic-based approach, i.e., it keeps records on the frequency of the paths used. Especially in highly dynamic networks, the SBR is more efficient in terms of reliability and reduced overhearing than the OLSR due to the utilization of this statistical data (cf. [97]). A more detailed analysis on the performance issues of routing protocols is outlined for instance in [96]. Nevertheless, the implementation and analysis of a routing service based on the EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocol as mentioned within this section remains a subject of future work.

8.2.3 Distributed Data Management

As mentioned in Definition 2.6, sensor nodes are not fail safe but rather error-prone. Hence, it may be wise to distribute (an aggregation of) sensor data or to share other important information, which is required to perform the common task of the WSN application. For instance, the Flooding Time-Synchronization Protocol [111] is based on a certain network procedure, which "floods" the network with specific data to enforce a specific situation of the network. According to Definition 2.6, a sensor node is severely limited in memory. Therefore, a sensor node may not be able to store all relevant data locally. Apart, sensor nodes tend to be error-prone and it may be reasonable to store important data redundantly distributed across (parts of) the network instead of in a single node. Both scenarios call for distributed *data management*, i.e., a sophisticated service to spread important data within the network.

One example for such a distributed data management protocol is provided in [74]: The proposed protocol implements a low-complexity greedy mechanism to distribute and also to replicate particular sensor data. The basis of this mechanism is the periodical broadcast of each node's so-called *memory advertisement message*. This message contains the currently available memory space of the sending node as well as the memory status of the sender's one-hop neighbors. With it, this data replication protocol increases the robustness of the Wireless Sensor Network against node failure or memory shortage. Notably, this protocol was evaluated empirically by real-world testbeds consisting of 78 sensor nodes within the SensLAB environment (cf. [35]).

Another example for such a data management protocol is the *Distributive Cluster Method* (DCM) in [41]. This protocol is beacon-based, i.e., it shares important data with nearby nodes by periodic messages. A more fault-tolerant variant of the DCM is introduced in [63][2]. The

[2] Diploma thesis conducted in conjunction with this work.

Local Fault-Tolerant Data Management (LFDM) approach implements the XOR operation to achieve a high reliability with little memory: On the one hand, each node broadcasts (e.g., as attachment to the corresponding firing packet) its data. On the other hand, each node combines the data received within one period utilizing logical operations. This allows the recreation of data of any one-hop neighbor in case of a node failure.

Once again, the periodic firing of our MAC protocol in conjunction with its exchange of neighbor information may lend support here. Nevertheless, the implementation and analysis of a data management service, e.g., the LFDM, based on the EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocols as mentioned within this section, remains a subject of future work.

Chapter 9

Summary and Conclusion

Abstract

This final chapter summarizes the outcome and the results of this work. In this regard, Section 9.1 first recapitulates the scientific contribution of this work and of our self-organizing MAC protocol for arbitrary multi-hop topologies. Finally, we conclude this work in Section 9.2.

9.1 Summary

The main focus of this work was on the design of a self-organizing multi-hop MAC protocol for arbitrary topology dynamics. In this regard, Part I specifies basic terms and essential definitions. As already mentioned in the introduction in Chapter 1, our main interest is in the analysis of the protocol's convergence behavior. For this purpose, we not only applied a sensor node framework in terms of real-world testbeds made up of different real sensor nodes, but we also developed a specific simulation framework. This framework meets our demands and supports the analysis of simulation scenarios from modeling and simulation to the *ex post facto* evaluation.

However, the core concepts of this work are addressed in Part II: Chapter 3 introduces the biologically inspired primitive of desynchronization as basis for our protocol. This primitive is the starting point for different approaches to establish a MAC protocol for Wireless Sensor Networks as described in Chapter 4. In fact, most of these protocols are applicable for single-hop WSNs only. Without doubt, multi-hop topologies are significantly more complex than single-hop topologies, e.g., due to the hidden terminal problem – to give but one example. Anyway, a MAC protocol which works well in arbitrary multi-hop topologies was seemingly missing.

In this context, we utilized the physically inspired proof of an elastic resilience model to prove the convergence of the midpoint approach for multi-hop topologies. As a result, the midpoint approach showed great promise to facilitate the development of a self-organizing MAC protocol, which also works well in arbitrary multi-hop topologies. Hence, based on this midpoint approach we developed the EXTENDED-DESYNC protocol in Chapter 5. In particular, we introduced the phase shift propagation approach to solve the hidden terminal problem. Here, each sender broadcasts information about the phase shifts of its one-hop neighbors. This allows each receiver to create its constraint graph and enables the EXTENDED-DESYNC protocol to be the first self-organizing MAC protocol, which is based on the primitive of desynchronization and, at the same time, supports Wireless Sensor Networks in arbitrary multi-hop topologies.

Nevertheless, the installation of the phase shift propagation enhancement is not sufficient to cope with any stale information related to the communication process: The reception of information about two-hop neighbors is deferred by about one period when utilizing the phase shift propagation. In short, the additional but essential exchange of neighbor information intensifies the stale information problem in multi-hop topologies as a side-effect. For this purpose, we improved our EXTENDED-DESYNC protocol to the EXTENDED-DESYNC$^+$ protocol in Chapter 6: In particular, we introduced the refractory threshold, which allows a sensor node to pause the adaptation process of a node's next time of firing of the underlying midpoint approach. This means that information about a neighbor may remain valid for another period, and thus may prevent additional "stale" information. As a result, this probabilistic protocol parameter may diminish the otherwise fluctuating behavior of the sensor nodes.

Our proposals from Part II are evaluated in Part III: In Chapter 7, we analyze several aspects of our protocols. For this purpose, we utilize real-world testbeds as well as our simulation framework as introduced in Chapter 2. As a result, several experiments and simulations do confirm that our protocols do meet our demands from Section 1.2. Nevertheless, there might be room for improvement. Potential enhancements regarding the key protocol parameters as well as possible add-ons (in terms of additional network services) are discussed in Chapter 8.

9.2 Conclusion

This work presents two self-organizing multi-hop MAC protocols for arbitrary topology dynamics, namely the EXTENDED-DESYNC and EXTENDED-DESYNC$^+$ protocol. The basis for this communication protocol is the biologically inspired primitive of desynchronization. We evaluated this primitive theoretically. For instance, we proved its convergence for the midpoint approach. Based on this midpoint approach, we developed the EXTENDED-DESYNC protocol to operate in multi-hop topologies. In this regard, we solved the hidden terminal problem by introducing the phase shift propagation. Besides, we also identified the stale information problem. For this problem, we installed the refractory threshold and made the resulting EXTENDED-DESYNC$^+$ protocol more resistant to stale information. Eventually, we developed for the communication within Wireless Sensor Networks a lightweight and applicable as well as scalable MAC protocol, which implements a self-organizing and dynamic TDMA schedule.

Their characteristics predestine our self-organizing protocols, namely EXTENDED-DESYNC and EXTENDED-DESYNC$^+$, to be used for various applications in highly dynamic networks. This includes not only arbitrary multi-hop topologies in Wireless Sensor Networks, and Wireless Networks in general. In addition, our protocols also support networks with mobile devices and can cope with malfunctioning network devices up to a certain extent. Moreover, the discussion on potential add-ons in Section 8.2 gives an outlook on the protocol's other capabilities, like time synchronization, routing, and data management.

Part IV

Lists and Indexes

> Wer am Ende ist, kann von vorn' anfangen, denn das Ende ist der Anfang von der anderen Seite
>
> *Karl Valentin*

Bibliography

Whenever possible, this bibliography avoids the declaration of an *Uniform Resource Locator* (URL) due to its transient nature. When an URL is essential for a citation and thus could not be avoided, the availability has been checked on August 4th, 2017.

[1] Rana E. Ahmed. Efficient pseudo-random number generators for wireless sensor networks. In *IEEE 59th International Midwest Symposium on Circuits and Systems*, MWSCAS 2016, pages 1–4, October 2016. doi: 10.1109/MWSCAS.2016.7869989.

[2] Ikkyu Aihara, Shunsuke Horai, Hiroyuki Kitahata, Kazuyuki Aihara, and Kenichi Yoshikawa. Dynamical Calling Behavior Experimentally Observed in Japanese Tree Frogs (Hyla japonica). *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E90-A (10):2154–2161, October 2007. ISSN 0916-8508. doi: 10.1093/ietfec/e90-a.10.2154.

[3] Ikkyu Aihara, Hiroyuki Kitahata, Kenichi Yoshikawa, and Kazuyuki Aihara. Mathematical modeling of frog's calling behavior and its possible application to artificial life and robotics. *Artificial Life and Robotics*, 12:29–32, 2008. ISSN 1433-5298. doi: 10.1007/s10015-007-0436-x.

[4] Kemal Akkaya and Mohamed F. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, May 2005. doi: 10.1016/j.adhoc.2003.09.010.

[5] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, October 2004. ISSN 15708705. doi: 10.1016/j.adhoc.2004.04.003.

[6] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, March 2002.

[7] Davide Anguita, Davide Brizzolara, and Giancarlo Parodi. *Prospects and Problems of Optical Diffuse Wireless Communication for Underwater Wireless Sensor Networks*, chapter 12, pages 205–230. Volume edited of Merrett and Tan [116], 2010. ISBN 978-953-307-321-7. doi: 10.5772/14472.

[8] Annam Thomas Antu and Paul Varghese. Random Number Generation Methods a Survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6(1):556–559, January 2016. ISSN 2277-128X. doi: 10.23956/ijarcsse.

[9] Alex Arenas, Albert Díaz Guilera, Jürgen Kurths, Yamir Moreno, and Changsong Zhou. Synchronization in complex networks. *Physics Reports*, 469(3):93–153, 12. December 2008. ISSN 0370-1573. doi: 10.1016/j.physrep.2008.09.002.

[10] AT45DB161B. *16-megabit 2.5-volt Only or 2.7-volt Only DataFlash AT45DB161B*. Atmel Corp., San Jose, CA (USA), October 2004.

[11] Ravi Bagree, Vishwas Raj Jain, Aman Kumar, and Prabhat Ranjan. TigerCENSE: Wireless Image Sensor Network to Monitor Tiger Movement. In Pedro J. Marron, Thiemo Voigt, Peter Corke, and Luca Mottola, editors, *4th International Workshop on Real-World Wireless Sensor Networks*, REALWSN 2010, pages 13–24. Springer, Berlin, Heidelberg, December 2010. ISBN 978-3-642-17520-6. doi: 10.1007/978-3-642-17520-6_2.

[12] Fabian Barthel. Regulierung des Overheads eines selbst-organisierenden MAC Protokolls durch Sendestärkenadaption. Bachelorarbeit, Universität Würzburg, 21. September 2013.

[13] Marcel Baunach. Ghost: Software and Configuration Distribution for Wireless Sensor/Actor Networks. In Hartmut Ritter, Kirsten Terfloth, Georg Wittenburg, and Jochen Schiller, editors, *7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 81–84, Berlin, Germany, September 2008. Freie Universität Berlin, Institute of Computer Science. Technical Report B 08-12.

[14] Marcel Baunach. Speed, Reliability and Energy Efficiency of HashSlot Communication in WSN Based Localization Systems. In Roberto Verdone, editor, *5th European Conference on Wireless Sensor Networks*, EWSN 2008, pages 74–89, Berlin, Heidelberg, 2008. Springer. ISBN 978-3-540-77690-1. doi: 10.1007/978-3-540-77690-1_5.

[15] Marcel Baunach. Dynamic Hinting: Real-Time Resource Management in Wireless Sensor/Actor Networks. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA 2009, pages 31–40. IEEE Computer Society, 2009. ISBN 978-0-7695-3787-0. doi: 10.1109/RTCSA.2009.65.

[16] Marcel Baunach. Dynamic Hinting: Collaborative Real-Time Resource Management for Reactive Embedded Systems. *Journal of Systems Architecture (JSA)*, 57:799–814, October 2011. ISSN 1383-7621. doi: 10.1016/j.sysarc.2011.07.001.

[17] Marcel Baunach. Towards Collaborative Resource Sharing under Real-Time Conditions in Multitasking and Multicore Environments. In *17th IEEE International Conference on Emerging Technologies Factory Automation*, ETFA 2012, pages 1–9. IEEE Computer Society, September 2012. ISBN 978-1-4673-4735-8. doi: 10.1109/ETFA.2012.6489587.

[18] Marcel Baunach. *Advances in Distributed Real-Time Sensor/Actuator Systems Operation – Operating Systems, Communication, and Application Design Concepts*. Dissertation, Institut für Informatik, Universität Würzburg, 30. October 2012.

[19] Marcel Baunach. Handling Time and Reactivity for Synchronization and Clock Drift Calculation in Wireless Sensor/Actuator Networks. In Marten van Sinderen, Octavian Postolache, César Benavente-Peces, and Ali Falah, editors, *3rd International Conference on Sensor Networks*, SENSORNETS 2014, pages 63–72, Lisbon, Portugal, January 2014. SciTePress. ISBN 978-989-758-001-7.

[20] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. SNoW5: A versatile ultra low power modular node for wireless ad hoc sensor networking. In Pedro José Marrón, editor, *5. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 55–59, Stuttgart, 17.–18. July 2006. Institut für Parallele und Verteilte Systeme.

[21] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. Beyond Theory: Development of a Real World Localization Application as Low Power WSN. In *32nd IEEE Conference on Local Computer Networks*, LCN 2007, pages 872–884, Washington, DC, USA, October 2007. IEEE Computer Society. ISBN 0-7695-3000-1. doi: 10.1109/LCN.2007.38.

[22] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. A Method for Self-Organizing Communication in WSN Based Localization Systems: HashSlot. In *32nd IEEE Conference on Local Computer Networks*, LCN 2007, pages 825–832, Washington, DC, USA, October 2007. IEEE Computer Society. ISBN 0-7695-3000-1. doi: 10.1109/LCN.2007.13. SenseApp 2007, Dublin (Ireland).

[23] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. SNoW5: a modular platform for sophisticated real-time wireless sensor networking. Technical Report 399, Institut für Informatik, Universität Würzburg, January 2007.

[24] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. Introduction to a Small Modular Adept Real-Time Operating System. In Distributed Systems Group, editor, *6. GI/ITG KuVS Fachgespräch Sensornetzwerke*, pages 1–4, Aachen, 16.–17. July 2007. RWTH Aachen University.

[25] Marcel Baunach, Reiner Kolla, and Clemens Mühlberger. SNoW Bat: A high precise WSN based location system. Technical Report 424, Institut für Informatik, Universität Würzburg, May 2007.

[26] Marcel Baunach, Clemens Mühlberger, Christian Appold, Martin Schröder, and Florian Füller. Analysis of Radio Signal Parameters for Calibrating RSSI Localization Systems. Technical Report 455, Institut für Informatik, Universität Würzburg, March 2009.

[27] Luca Bencini, Davide Di Palma, Giovanni Collodi, Antonio Manes, and Gianfranco Manes. *Wireless Sensor Networks for On-Field Agricultural Management Process*, chapter 2, pages 17–34. Volume edited of Merrett and Tan [116], 2010. ISBN 978-953-307-321-7. doi: 10.5772/13001.

[28] Jan Beutel, Oliver Kasten, Friedemann Mattern, Kay Römer, Frank Siegemund, and Lothar Thiele. Prototyping Wireless Sensor Network Applications with BTnodes. In Holger Karl, Adam Wolisz, and Andreas Willig, editors, *1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338, Berlin, January 2004. Springer. ISBN 978-3-540-24606-0. doi: 10.1007/978-3-540-24606-0_22.

[29] Bela Bollobás. *Modern Graph Theory*. Graduate Texts in Mathematics. Springer, July 1998. ISBN 0-387-98488-7.

[30] Nuttapon Boonpinon and Attawith Sudsang. Heterogeneity Driven Circular Formation. In *IEEE International Conference on Robotics and Biomimetics*, ROBIO 2006, pages 971–976. IEEE Computer Society, 2006. ISBN 1-4244-0570-X. doi: 10.1109/ROBIO.2006.340360.

[31] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in Network Simulation. *Computer*, 33(5):59–67, May 2000. ISSN 0018-9162. doi: 10.1109/2.841785.

[32] Bernhard Buchli, Felix Sutton, and Jan Beutel. GPS-Equipped wireless sensor network node for high-accuracy positioning applications. In *9th European Conference on Wireless Sensor Networks (EWSN 2012)*, volume 7158 of *Lecture Notes in Computer Science*, pages 179–195, Berlin, Heidelberg, February 2012. Springer. ISBN 978-3-642-28168-6. doi: 10.1007/978-3-642-28169-3_12.

[33] Dujdow Buranapanichkit. *Multichannel Distributed Coordination for Wireless Sensor Networks: Convergence Delay and Energy Consumption Aspects*. Dissertation, Department of Electronic and Electrical Engineering, University College London, UK, August 2013.

[34] Dujdow Buranapanichkit, Nikos Deligiannis, and Yiannis Andreopoulos. Convergence of Desynchronization Primitives in Wireless Sensor Networks: A Stochastic Modeling Approach. *IEEE Transactions on Signal Processing*, 63(1):221–233, Jan 2015. ISSN 1053-587X. doi: 10.1109/TSP.2014.2369003.

[35] Clément Burin des Rosiers, Guillaume Chelius, Eric Fleury, Antoine Fraboulet, Antoine Gallais, Nathalie Mitton, and Thomas Noël. SensLAB - Very Large Scale Open Wireless Sensor Network Testbed. In Thanasis Korakis, Hongbin Li, Phuoc Tran-Gia, and Hong-Shik Park, editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 90 of *TridentCom 2011*, pages 239–254. Springer, Berlin, Heidelberg, April 2012. ISBN 978-3-642-29273-6. doi: 10.1007/978-3-642-29273-6_19.

[36] Marcus Chang. Power Efficient Duty-cycling with Ultra Low-power Receivers. Masterthesis, University of Copenhagen, Denmark, June 2006.

[37] Chien-Ying Chen, Yu-Ting Chen, Yi-Hsuan Tu, Shun-Yao Yang, and Pai H. Chou. EcoSpire: An Application Development Kit for an Ultra-Compact Wireless Sensing System. *IEEE Embedded Systems Letters*, 1(3):65–68, October 2009. ISSN 1943-0663. doi: 10.1109/LES.2009.2037984.

[38] Supasate Choochaisri. *A Physicomimetics Desynchronization Algorithm without Global Time Knowledge for Wireless Sensor Networks*. Dissertation, Faculty of Engineering, Chulalongkorn University, Thailand, 11. May 2012.

[39] Supasate Choochaisri, Kittipat Apicharttrisorn, Kittiporn Korprasertthaworn, Pongpakdi Taechalertpaisarn, and Chalermek Intanagonwiwat. Desynchronization with an Artificial Force Field for Wireless Networks. *SIGCOMM Comput. Commun. Rev.*, 42(2):7–15, April 2012. ISSN 0146-4833. doi: 10.1145/2185376.2185378.

[40] Thomas Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, RFC Editor, October 2003.

[41] Alexander Coers. *Verfahren zur redundanten und distributiven Datenverarbeitung in drahtlosen Sensornetzen*. Fraunhofer IRB Verlag, 2007. ISBN 978-3-8167-7393-1.

[42] Sinem Coleri Ergen and Pravin Varaiya. PEDAMACS: power efficient and delay aware medium access protocol for sensor networks. *IEEE Transactions on Mobile Computing*, 5(7):920–930, July 2006. ISSN 1536-1233. doi: 10.1109/TMC.2006.100.

[43] MICA2. *MICA2 Wireless Measurement System*. Crossbow Technology Inc., San Jose (USA), 2005.

[44] TelosB. *TelosB Mote Platform*. Crossbow Technology Inc., San Jose (USA), 2006.

[45] Imote2. *Imote2 High-Performance Wireless Sensor Network Node*. Crossbow Technology Inc., San Jose (USA), 2007.

[46] MICAz. *MICAz Wireless Measurement System*. Crossbow Technology Inc., San Jose (USA), 2007.

[47] Hui Dai and Richard Han. TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8 (1):125–139, January 2004. ISSN 1559-1662. doi: 10.1145/980159.980173.

[48] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of Wireless Sensor Networks – Theory and Practice*. Wireless Communications and Mobile Computing. Wiley, July 2010. ISBN 978-0-470-99765-9.

[49] Julius Degesys and Radhika Nagpal. Towards Desynchronization of Multi-hop Topologies. In *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO 2008, pages 129–138, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3404-6. doi: 10.1109/SASO.2008.70.

[50] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks. In Tarek F. Abdelzaher, Leonidas J. Guibas, and Matt Welsh, editors, *6th International Conference on Information Processing in Sensor Networks*, IPSN 2007, pages 11–20, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-638-7. doi: 10.1145/1236360.1236363.

[51] Julius Degesys, Prithwish Basu, and Jason Redi. Synchronization of Strongly Pulse-Coupled Oscillators with Refractory Periods and Random Medium Access. In *ACM Symposium on Applied Computing*, SAC 2008, pages 1976–1980, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-753-7. doi: 10.1145/1363686.1364164.

[52] Hans Dodel and Dieter Häupler. *Satellitennavigation*. Springer, 2nd edition, 2010. ISBN 978-3-540-79443-1. doi: 10.1007/978-3-540-79444-8.

[53] Johannes Dölfel. Development and Evaluation of a UMTS-based Architecture for Vehicular Local Danger Warning. Diplomarbeit, University of Würzburg, Würzburg, 17. August 2007.

[54] Jingyi Dong. Implementierung eines Schallquelle Ortungssystems mit SNoW5 Sensorknoten. Bachelorarbeit, Universität Würzburg, 02. January 2012.

[55] Falko Dressler. *Self-Organization in Sensor and Actor Networks*. John Wiley & Sons, December 2007. ISBN 978-0-470-02820-9. doi: 10.1002/9780470724460.

[56] Frederick Ducatelle, Gianni A. Di Caro, and Luca M. Gambardella. Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intelligence*, 4(3):173–198, September 2010. ISSN 1935-3820. doi: 10.1007/s11721-010-0040-x.

[57] Hassan Echoukairi, Khalid Bourgba, and Mohammed Ouzzif. A Survey on Flat Routing Protocols in Wireless Sensor Networks. In Essaïd Sabir, Hicham Medromi, and Mohamed Sadik, editors, *Advances in Ubiquitous Networking: Proceedings of the UNet'15*, volume 366 of *Lecture Notes in Electrical Engineering (LNEE)*, pages 311–324. Springer, Singapore, 2016. ISBN 978-981-287-990-5. doi: 10.1007/978-981-287-990-5_25.

[58] Amre El Hoiydi and Jean-Dominique Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In Sotiris E. Nikoletseas and José D. P. Rolim, editors, *1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, ALGOSENSORS 2004, pages 18–31. Springer, Berlin, Heidelberg, 16. July 2004. ISBN 978-3-540-27820-7. doi: 10.1007/978-3-540-27820-7_4.

[59] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *5th Symposium on Operating Systems Design and Implementation*, volume 36 of *OSDI 2002*, pages 147–163, Boston, MA, USA, December 2002. ACM. doi: 10.1145/844128.844143.

[60] Andreas Engel. SNoW-RA: Reap and Allot – Energieneutraler Betrieb des SNoW5-Sensorknotens durch Gewinnen und Verteilen von Solarenergie. Diplomarbeit, Universität Würzburg, 5. July 2009.

[61] EstiNet Technologies Inc. *The GUI User Manual for the EstiNet 9.0 Network Simulator*, 3. August 2015.

[62] EstiNet Technologies Inc. EstiNet Technologies, 2016. URL `http://www.estinet.com/ns/`.

[63] Patrick Fakesch. Methoden zur fehlertoleranten Datenhaltung in Sensornetzen. Diplomarbeit, Universität Würzburg, 31. August 2010.

[64] Kevin Fall, Kannan Varadhan, and the VINT project. *The ns Manual*, 01. November 2011. URL `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`.

[65] Paul M. Fishbane, Stephen G. Gasiorowicz, and Stephen T. Thornton. *Physics for Scientists and Engineers*. Pearson Prentice Hall, 3rd edition, 2005. ISBN 978-0-13-191182-6.

[66] *s-net® – Wireless Sensor Networks*. Fraunhofer Institute for Integrated Circuits (IIS), September 2011. URL `http://www.s-net-info.com`.

[67] FU Berlin. ScatterWeb – Embedded Sensor Board (ESB), 2005. URL `http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/ScatterWeb/`.

[68] Florian Füller. Entwicklung eines Protokolls zu einer RSS-Lokalisationsinfrastruktur in drahtlosen Sensornetzen. Diplomarbeit, Universität Würzburg, 20. February 2009.

[69] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync Protocol for Sensor Networks. In *1st International Conference on Embedded Networked Sensor Systems*, SenSys 2003, pages 138–149, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-707-9. doi: 10.1145/958491.958508.

[70] Saurabh Ganeriwal, Jeremy Elson, and Mani B. Srivastava. Time Synchronization. In Nirupama Bulusu and Sanjay Jha, editors, *Wireless Sensor Networks - A Systems Perspective*, MEMS, chapter 5, pages 59–74. Artech House, 2005. ISBN 978-1-58053-867-1.

[71] James E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer-Verlag, 2nd edition, 2003. ISBN 978-0-387-00178-4. doi: 10.1007/b97336.

[72] Douglas C. Giancoli. *Physik – Lehr- und Arbeitsbuch*. Pearson Studium, 3rd ext. edition, October 2009. ISBN 978-3-8689-4023-7.

[73] Alessandro Giusti, Amy L. Murphy, and Gian Pietro Picco. Decentralized Scattering of Wake-up Times in Wireless Sensor Networks. In *4th European Conference on Wireless Sensor Networks*, EWSN 2007, pages 245–260, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-69829-6. doi: 10.1007/978-3-540-69830-2_16.

[74] Pietro Gonizzi, Gianluigi Ferrari, Vincent Gay, and Jeremie Leguay. REDUNDANT DISTRIBUTED DATA STORAGE - Experimentation with the SensLab Testbed. In Marten van Sinderen, Octavian Postolache, and César Benavente-Peces, editors, *1st International Conference on Sensor Networks*, SENSORNETS 2012, pages 15–23, Rome, Italy, February 2012. SciTePress. ISBN 978-989-8565-01-3. doi: 10.5220/0003803900150023.

[75] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification – Java SE 7 Edition*. Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065, USA, 28. February 2013.

[76] Harry Gros desormeaux, Philippe Hunel, and Nicolas Vidot. *Wildlife Assessment Using Wireless Sensor Networks*, chapter 3, pages 35–49. Volume edited of Merrett and Tan [116], 2010. ISBN 978-953-307-321-7. doi: 10.5772/13812.

[77] G.P. Halkes and K.G. Langendoen. Crankshaft: An Energy-Efficient MAC-Protocol for Dense Wireless Sensor Networks. In *4th European Conference on Wireless Sensor Networks*, EWSN 2007, pages 228–244, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-69829-6. doi: 10.1007/978-3-540-69830-2_15.

[78] Ulrich Hedtstück. *Simulation diskreter Prozesse*. Springer Verlag, 2013. ISBN 978-3-642-34870-9. doi: 10.1007/978-3-642-34871-6.

[79] John Heidemann, Milica Stojanovic, and Michele Zorzi. Underwater sensor networks: applications, advances and challenges. *Philosophical Transactions of The Royal Society A*, 370(1958):158–175, January 2012. doi: 10.1098/rsta.2011.0214.

[80] Hugo Hernández and Christian Blum. Implementing a Model of Japanese Tree Frogs' Calling Behavior in Sensor Networks: a Study of Possible Improvements. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO 2011, pages 615–622, New York, NY (USA), July 2011. ACM. ISBN 978-1-4503-0690-4. doi: 10.1145/2001858.2002057.

[81] Hugo Hernández and Christian Blum. Distributed graph coloring: an approach based on the calling behavior of Japanese tree frogs. *Swarm Intelligence*, 6(2):117–150, 2012. ISSN 1935-3812. doi: 10.1007/s11721-012-0067-2.

[82] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System Architecture Directions for Networked Sensors. *SIGPLAN Not.*, 35(11):93–104, November 2000. ISSN 0362-1340. doi: 10.1145/356989.356998.

[83] Bernhard Hofmann Wellenhof, Herbert Lichtenegger, and Elmar Wasle. *GNSS Global Navigation Satellite Systems – GPS, GLONASS, Galileo, and more*. Springer, 2008. ISBN 978-3-211-73012-6. doi: 10.1007/978-3-211-73017-1.

[84] Yao-Win Hong and Anna Scaglione. A Scalable Synchronization Protocol For Large Scale Sensor Networks And Its Applications. *IEEE Journal on Selected Areas in Communications*, 23(5):1085–1099, May 2005. ISSN 0733-8716. doi: 10.1109/JSAC.2005.845418.

[85] Jeonghwan Hwang, Changsun Shin, and Hyun Yoe. Study on an Agricultural Environment Monitoring Server System using Wireless Sensor Networks. *Sensors*, 10(12): 11189–11211, 2010. ISSN 1424-8220. doi: 10.3390/s101211189.

[86] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* (IEEE Std 802.11-2007). IEEE Computer Society, New York, NY 10016-5997, USA, June 2007.

[87] *ITU-T Recommendation X.200*. International Telecommunication Union (ITU), July 1994.

[88] *Quartz Crystal Specification – HC49/3H*. IQD Frequency Products, July 2016.

[89] Eugene M. Izhikevich. Weakly pulse-coupled oscillators, FM interactions, synchronization, and oscillatory associative memory. *IEEE Transactions on Neural Networks*, 10(3):508–526, May 1999. ISSN 1045-9227. doi: 10.1109/72.761708.

[90] Xiaofan Jiang, Joseph Polastre, and David Culler. Perpetual Environmentally Powered Sensor Networks. In *4th International Symposium on Information Processing in Sensor Networks*, IPSN 2005, pages 463–468, Piscataway, NJ, USA, April 2005. IEEE Press. ISBN 0-7803-9202-7. doi: 10.1109/IPSN.2005.1440974.

[91] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 96–107, New York, NY, USA, December 2002. ACM. ISBN 1-58113-574-2. doi: 10.1145/605397.605408.

[92] Hui Kang and Jennifer L. Wong. A Localized Multi-Hop Desynchronization Algorithm for Wireless Sensor Networks. In *INFOCOM 2009, IEEE*, pages 2906–2910, Rio de Janeiro, Brazil, April 2009. IEEE Computer Society. doi: 10.1109/INFCOM.2009.5062256.

[93] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005. ISBN 978-0-470-09510-2.

[94] Phil Karn. MACA - A new channel access method for packet radio. In *Computer Networking Conference*, pages 134–140, London, ON, Canada, September 1990.

[95] Martin Keupp. Experimentelle Evaluation der Robustheit eines selbst-organisierenden MAC Protokolls für Multi-Hop Topologien. Bachelorarbeit, Universität Würzburg, 24. July 2012.

[96] Alexander Klein. *Performance Issues of MAC and Routing Protocols in Wireless Sensor Networks*. Dissertation, Institut für Informatik, Universität Würzburg, 25. May 2010.

[97] Alexander Klein and Phuoc Tran Gia. A Statistic-Based Approach towards Routing in Mesh Networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–6. IEEE Computer Society, October 2007. doi: 10.1109/MOBHOC.2007.4428722.

[98] Franziska Klügl. *Multiagentensimulation. Konzepte, Werkzeuge, Anwendung*. Agententechnologie. Addison-Wesley Verlag, 2001. ISBN 978-3-8273-1790-2.

[99] Keiji Konishi and Hideki Kokame. Synchronization of pulse-coupled oscillators with a refractory period and frequency distribution for a wireless sensor network. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):033132, 2008. doi: 10.1063/1.2970103.

[100] Hermann Kopetz and Wilhelm Ochsenreiter. Clock Synchronization in Distributed Real-time Systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987. ISSN 0018-9340. doi: 10.1109/TC.1987.5009516.

[101] Sven Oliver Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Leitfäden der Informatik. Vieweg + Teubner, 3rd edition, 2012. ISBN 978-3-8348-1849-2. doi: 10.1007/978-3-8348-2264-2.

[102] Horst Kuchling. *Taschenbuch der Physik*. Hanser, 20th edition, November 2010. ISBN 978-3-446-42457-9.

[103] Jeevan Kumar, Sachin Tripathi, and Rajesh Kumar Tiwari. A Survey on Routing Protocols for Wireless Sensor Networks Using Swarm Intelligence. *International Journal of Internet Technology and Secured Transactions*, 6(2):79–102, January 2016. ISSN 1748-569X. doi: 10.1504/IJITST.2016.078574.

[104] Averill M. Law and David M. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd edition, 1999. ISBN 978-0-07-059292-6.

[105] Pierre L'Ecuyer. Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation*, 68(225):249–260, January 1999. ISSN 1088-6842. doi: 10.1090/S0025-5718-99-00996-5.

[106] WaspMote. *Waspmote Datasheet*. Libelium Comunicaciones Distribuidas S.L., Zaragoza (Spain), v7.0 edition, oct 2016.

[107] Jesus Antonio Llor Sirvent and Manuel Perez Malumbres. *Modelling Underwater Wireless Sensor Networks*, chapter 11, pages 185–203. Volume edited of Merrett and Tan [116], 2010. ISBN 978-953-307-321-7. doi: 10.5772/13144.

[108] Dennis Lucarelli and I-Jeng Wang. Decentralized Synchronization Protocols with Nearest Neighbor Communication. In John A. Stankovic, Anish Arora, and Ramesh Govindan, editors, *2nd International Conference on Embedded Networked Sensor Systems*, SenSys 2004, pages 62–68, New York, NY, USA, November 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031503.

[109] Gerald Lutter. *Filesystem für YAOS auf einem MSP430 mit AT54DB161B*. Department of Computer Engineering, University of Wuerzburg, Würzburg, Germany, 2006.

[110] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA 2002, pages 88–97, New York, NY, USA, 2002. ACM. ISBN 1-58113-589-0. doi: 10.1145/570738.570751.

[111] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The Flooding Time Synchronization Protocol. In John A. Stankovic, Anish Arora, and Ramesh Govindan, editors, *2nd International Conference on Embedded Networked Sensor Systems*, SenSys 2004, pages 39–49, New York, NY, USA, November 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031501.

[112] Rudolf Mathar and Jürgen Mattfeldt. Pulse-Coupled Decentral Synchronization. *SIAM Journal on Applied Mathematics*, 56(4):1094–1106, 1996. ISSN 0036-1399. doi: 10.1137/S0036139994278135.

[113] Steven McCanne and Sally Floyd. The Network Simulator – ns-2, 2011. URL `http://www.isi.edu/nsnam/ns/`.

[114] Lennart Meier, Philipp Blum, and Lothar Thiele. Internal Synchronization of Drift-Constraint Clocks in Ad-Hoc Sensor Networks. In *5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc 2004, pages 90–97, New York, NY, USA, 2004. ACM. ISBN 1-58113-849-0. doi: 10.1145/989459.989471.

[115] Rabah Meraihi, Sidi-Mohammed Senouci, Djamal-Eddine Meddour, and Moez Jerbi. Vehicle-to-Vehicle Communications: Applications and Perspectives. In Houda Labiod, editor, *Wireless Ad Hoc and Sensor Networks*, ISTE, chapter 12, pages 285–308. John Wiley & Sons, 2008. ISBN 978-1-84821-003-5.

[116] Geoff V. Merrett and Yen Kheng Tan, editors. *Wireless Sensor Networks: Application-Centric Design*, volume edited. InTech, 2010. ISBN 978-953-307-321-7. doi: 10.5772/658.

[117] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the Internet of Things (IoT). *IEEE Internet Initiative*, 1:1–86, 27. May 2015.

[118] Renato E. Mirollo and Steven H. Strogatz. Synchronization of Pulse-Coupled Biological Oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990. ISSN 0036-1399. doi: 10.1137/0150098.

[119] Arik Motskin, Tim Roughgarden, Primoz Skraba, and Leonidas J. Guibas. Lightweight Coloring and Desynchronization for Networks. In *INFOCOM 2009, IEEE*, pages 2383–2391, Rio de Janeiro, Brazil, April 2009. IEEE Computer Society. doi: 10.1109/INFCOM.2009.5062165.

[120] Clemens Mühlberger. Vergleich von Ereignis-basierter und Takt-gesteuerter Simulation in Multiagentensystemen. Diplomarbeit, Universität Würzburg, Würzburg, 27. January 2005.

[121] Clemens Mühlberger. Energetic and Temporal Analysis of a Desynchronized TDMA Protocol for WSNs. In Institut für Telematik, editor, *8. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 59–62, Hamburg, Germany, August 2009. Technische Universität Hamburg-Harburg, Institute of Telematics.

[122] Clemens Mühlberger. Desynchronization in Multi-Hop Topologies: A Challenge. In Reiner Kolla, editor, *9. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 21–24, Würzburg, Germany, September 2010. Universität Würzburg, Institut für Informatik. urn:nbn:de:bvb:20-opus-51106.

[123] Clemens Mühlberger. Integrating Time-Stamped Synchronization into a Periodical MAC Protocol - Problems and Experiences. Technical Report 477, Institut für Informatik, Universität Würzburg, December 2010.

[124] Clemens Mühlberger. On the Improvement of a Self-Organized MAC Protocol for Multi-Hop Wireless Sensor Networks. In *11. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 7–10, Darmstadt, Germany, September 2012. Technische Universität Darmstadt, Fachbereich Informatik. SEEMOO-TR-2012-03.

[125] Clemens Mühlberger. Analyzing a Self-Organizing Multi-Hop Protocol: Ease of Simulations and Need for Real-World Tests. In *9th International Wireless Communications and Mobile Computing Conference*, IWCMC 2013, pages 1029–1034. IEEE Computer Society, 2013. ISBN 978-1-4673-2479-3. doi: 10.1109/IWCMC.2013.6583698.

[126] Clemens Mühlberger. On the Pitfalls of Desynchronization in Multi-hop Topologies. In Marten van Sinderen, Octavian Postolache, and César Benavente-Peces, editors, *2nd International Conference on Sensor Networks*, SENSORNETS 2013, pages 99–108, Barcelona, Spain, February 2013. SciTePress. ISBN 978-989-8565-45-7. doi: 10.5220/0004230900990108. Best Student Paper Award.

[127] Clemens Mühlberger and Marcel Baunach. Tab WoNS: Calibration Approach for WSN based Ultrasound Localization Systems. In Hartmut Ritter, Kirsten Terfloth, Georg Wittenburg, and Jochen Schiller, editors, *7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 41–44, Berlin, Germany, September 2008. Freie Universität Berlin, Institute of Computer Science. Technical Report B 08-12.

[128] Clemens Mühlberger and Reiner Kolla. Extended Desynchronization for Multi-Hop Topologies. Technical Report 460, Institut für Informatik, Universität Würzburg, July 2009.

[129] Clemens Mühlberger and Tobias Schäfer. SuperG: A Multi-Radio Architecture to interconnect multiple Wireless Sensor Networks. In Hannes Frey, editor, *10. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 13–16, Paderborn, Germany, September 2011. Universität Paderborn, Institut für Informatik. TR-RI-11-313.

[130] David Murray, Micheal Dixon, and Terry Koziniec. An Experimental Comparison of Routing Protocols in Multi Hop Ad Hoc Networks. In *Australasian Telecommunication Networks and Applications Conference (ATNAC)*, pages 159–164, October 2010. doi: 10.1109/ATNAC.2010.5680190.

[131] Akira Mutazono. Frog Call-Inspired Self-Organizing Transmission Scheduling Scheme for Wireless Sensor Networks. Masterthesis, Graduate School of Information Science Technology, Osaka University, February 2009.

[132] Akira Mutazono, Masashi Sugano, and Masayuki Murata. Frog Call-Inspired Self-Organizing Anti-Phase Synchronization for Wireless Sensor Networks. In *2nd International Workshop on Nonlinear Dynamics and Synchronization*, INDS 2009, pages 81–88, Klagenfurt (Austria), July 2009.

[133] Akira Mutazono, Masashi Sugano, and Masayuki Murata. Self-organizing Anti-phase Synchronization Scheme for Sensor Networks Inspired by Frogs' Calling Behavior. *ISAST Transactions on Computers and Intelligent Systems*, 1(2):86–93, 2009. ISSN 1798-2448.

[134] Akira Mutazono, Masashi Sugano, and Masayuki Murata. Comparison of robustness of time synchronisation in sensor networks. *International Journal of Autonomous and Adaptive Communications Systems (IJAACS)*, 4(2):202–216, 2011. doi: 10.1504/IJAACS.2011.039724.

[135] Siri Namtvedt. *Design Note DN506 - GDO Pin Usage*. Texas Instruments Inc., Dallas, Texas (USA), October 2007.

[136] Bernd Neubig and Wolfgang Briese. *Das große Quarzkochbuch*. Franzis Verlag, 2nd edition, 2000. ISBN 978-3-7723-5853-1.

[137] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. ISBN 978-0-898712-95-7.

[138] Lars Niemann, Marcus Venzke, Christian Renner, and Volker Turau. Clock Synchronization of TinyOS-based Sensor Networks with DCF77. In Institut für Telematik, editor, *8. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 45–46, Hamburg, Germany, August 2009. Technische Universität Hamburg-Harburg, Institute of Telematics.

[139] OpenSim Ltd. OMNeT++, 2015. URL `https://omnetpp.org`.

[140] Luigi Palopoli, Roberto Passerone, Amy L. Murphy, Gian Pietro Picco, and Alessandro Giusti. Solving the Wake-Up Scattering Problem Optimally. In *6th European Conference on Wireless Sensor Networks*, EWSN 2009, pages 166–182, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-00223-6. doi: 10.1007/978-3-642-00224-3_11.

[141] Ankit Patel, Julius Degesys, and Radhika Nagpal. Desynchronization: The Theory of Self-Organizing Algorithms for Round-Robin Scheduling. In *1st International Conference on Self-Adaptive and Self-Organizing Systems*, SASO 2007, pages 87–96, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2906-2. doi: 10.1109/SASO.2007.17.

[142] Charles S. Peskin. *Mathematical Aspects of Heart Physiology*. Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, NY 10012, USA, 1975.

[143] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *2nd International Conference on Embedded Networked Sensor Systems*, SenSys 2004, pages 95–107, New York, NY, USA, November 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031508.

[144] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *4th International Conference on Information Processing in Sensor Networks*, IPSN/SPOTS 2005, 25.–27. April 2005.

[145] Oleksandr V. Popovych, Peter A. Tass, and Christian Hauptmann. Desynchronization (computational neuroscience). *Scholarpedia*, 6(10):1352, 2011. doi: 10.4249/scholarpedia.1352.

[146] Ramjee Prasad and Marina Ruggieri, editors. *Applied Satellite Navigation Using GPS, GALILEO, and Augmentation Systems*. mobile communications series. Artech House, 2005. ISBN 978-1-58053-814-2.

[147] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani Srivastava. Design Considerations for Solar Energy Harvesting Wireless Embedded Systems. In *4th International Symposium on Information Processing in Sensor Networks*, IPSN 2005, pages 457–462, Piscataway, NJ, USA, April 2005. IEEE Press. ISBN 0-7803-9202-7. doi: 10.1109/IPSN.2005.1440973.

[148] Venkatesh Rajendran, Katia Obraczka, and J.J. Garcia-Luna Aceves. Energy-efficient Collision-free Medium Access Control for Wireless Sensor Networks. In *1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 181–192, New York, NY, USA, November 2003. ACM. ISBN 1-58113-707-9. doi: 10.1145/958491.958513.

[149] Venkatesh Rajendran, Katia Obraczka, and J.J. Garcia-Luna Aceves. Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks. *Wireless Networks*, 12(1):63–78, February 2006. ISSN 1022-0038. doi: 10.1007/s11276-006-6151-z.

[150] *Renesas Starter Kit+ for SH7203 – User's Manual*. Renesas Electronics Corporation, Kanagawa (Japan), reg10j0060-0200 edition, 17. January 2009.

[151] *SH7203 Group – Hardware Manual*. Renesas Electronics Corporation, Kanagawa (Japan), rej09b0313-0300 edition, 28. September 2009.

[152] Tobias Reusing. Implementierung und Analyse eines WSN gestützten Lokalisationsalgorithmus auf Basis des SNoW5 Sensorknotens. Bachelorarbeit, Universität Würzburg, 18. June 2010.

[153] Ill-Keun Rhee, Jaehan Lee, Jangsub Kim, Erchin Serpedin, and Yik-Chung Wu. Clock Synchronization in Wireless Sensor Networks: An Overview. *Sensors*, 9(1):56–85, January 2009. ISSN 1424-822. doi: 10.3390/s90100056.

[154] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-MAC: a Hybrid MAC for Wireless Sensor Networks. In Jason Redi, Hari Balakrishnan, and Feng Zhao, editors, *3rd International Conference on Embedded Networked Sensor Systems*, SenSys 2005, pages 90–101, San Diego, CA, USA, November 2005. ACM. ISBN 1-59593-054-X. doi: 10.1145/1098918.1098929.

[155] S. W. Roberts. Control Chart Tests Based on Geometric Moving Averages. *Technometrics*, 1(3):239–250, August 1959. ISSN 0040-1706.

[156] Kay Römer. Time Synchronization in Ad Hoc Networks. In *2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc 2001, pages 173–182, New York, NY, USA, 2001. ACM. ISBN 1-58113-428-2. doi: 10.1145/501436.501440.

[157] Tobias Schäfer. Implementierung eines Multilayer Multiradio Gateways für drahtlose Sensornetze. Diplomarbeit, Universität Würzburg, 26. July 2011.

[158] Erchin Serpedin and Qasim M. Chaudhari. *Synchronization in Wireless Sensor Networks: Parameter Estimation, Peformance Benchmarks, and Protocols*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN 978-0-521-76442-1.

[159] Tossaphol Settawatcharawanit, Supasate Choochaisri, Chalermek Intanagonwiwat, and Kultida Rojviboonchai. V-DESYNC: Desynchronization for Beacon Broadcasting on Vehicular Networks. In *75th IEEE Vehicular Technology Conference*, VTC Spring 2012, pages 1–5, 2012. doi: 10.1109/VETECS.2012.6239966.

[160] Hans-Jürgen Siegert. *Simulation zeitdiskreter Systeme*. R. Oldenbourg, 1991. ISBN 978-3486218329.

[161] Osvaldo Simeone and Umberto Spagnolini. Distributed Time Synchronization in Wireless Sensor Networks with Coupled Discrete-Time Oscillators. *EURASIP Journal on Wireless Communications and Networking*, 2007(1):057054, March 2007. ISSN 1687-1499. doi: 10.1155/2007/57054.

[162] Farhan Simjee and Pai H. Chou. Everlast: Long-life, Supercapacitor-operated Wireless Sensor Node. In *International Symposium on Low Power Electronics and Design*, ISLPED 2006, pages 197–202, New York, NY, USA, October 2006. ACM. ISBN 1-59593-462-6. doi: 10.1145/1165573.1165619.

[163] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor Network-based Countersniper System. In *2nd International Conference on Embedded Networked Sensor Systems*, SenSys 2004, pages 1–12, New York, NY, USA, 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031497.

[164] Fikrit Sivrikaya and Bülent Yener. Time Synchronization. In Zheng and Jamalipour [200], chapter 9, pages 285–306. ISBN 978-0-470-16763-2.

[165] Michael Soffel and Ralf Langhans. *Space-Time Reference Systems*. Astronomy and Astrophysics Library. Springer, 2013. ISBN 978-3-642-30225-1. doi: 10.1007/978-3-642-30226-8.

[166] Kazem Sohraby, Daniel Minoli, and Taieb Znati. *Wireless Sensor Networks – Technology, Protocols, and Applications*. John Wiley & Sons, 2007. ISBN 978-0-471-74300-2.

[167] Daniel Stegmeier. Einfache Zeitsynchronisation in drahtlosen Sensornetzen. Diplomarbeit, Universität Würzburg, 23. November 2009.

[168] Stig Støa and Ilangko Balasingham. Periodic-MAC: Improving MAC Protocols for Biomedical Sensor Networks Through Implicit Synchronization. In Anthony N. Laskovski, editor, *Biomedical Engineering Trends in Electronics, Communications and Software*, chapter 26, pages 507–522. InTech, Rijeka, Croatia, January 2011. ISBN 978-953-307-475-7. doi: 10.5772/13223.

[169] Girts Strazdins, Atis Elsts, Krisjanis Nesenbergs, and Leo Selavo. Wireless Sensor Network Operating System Design Rules Based on Real-World Deployment Survey. *Journal of Sensor and Actuator Networks*, 2(3):509–556, 2013. ISSN 2224-2708. doi: 10.3390/jsan2030509.

[170] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press, 2000. ISBN 978-0-7382-0453-6.

[171] Steven H. Strogatz. *Sync: How Order Emerges From Chaos In the Universe, Nature, and Daily Life*. Hyperion Books, 2003. ISBN 978-0-7868-8721-7.

[172] Maneesha Sudheer. *Wireless Sensor Network for Disaster Monitoring*, chapter 3, pages 51–70. Volume edited of Merrett and Tan [116], 2010. ISBN 978-953-307-321-7. doi: 10.5772/13119.

[173] Pongpakdi Taechalertpaisarn, Supasate Choochaisri, and Chalermek Intanagonwiwat. An Orthodontics-Inspired Desynchronization Algorithm for Wireless Sensor Networks. In *13th IEEE International Conference on Communication Technology*, ICCT 2011, pages 631–636, Jinan, China, September 2011. doi: 10.1109/ICCT.2011.6157953.

[174] *MSP430x1xx Family User's Guide*. Texas Instruments Inc., Dallas, Texas (USA), 2006.

[175] *CC1100 Low-Power Sub-1 GHz RF Transceiver*. Texas Instruments Inc., Dallas, Texas (USA), December 2009.

[176] *CC2520 2.4 GHZ IEEE 802.15.4/ZIGBEE® RF TRANSCEIVER*. Texas Instruments Inc., Dallas, Texas (USA), December 2009.

[177] *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller*. Texas Instruments Inc., Dallas, Texas (USA), March 2011.

[178] *MSP430™ SoC With RF Core*. Texas Instruments Inc., Dallas, Texas (USA), September 2013.

[179] *CC430 Family User's Guide*. Texas Instruments Inc., Dallas, Texas (USA), January 2013.

[180] *eZ430-Chronos™ Development Tool User's Guide*. Texas Instruments Inc., Dallas, Texas (USA), January 2013.

[181] Paul A. Tipler and Gene Mosca. *Physik für Wissenschaftler und Ingenieure*. Spektrum Akademischer Verlag, 6th edition, August 2009. ISBN 978-3827419453.

[182] Fouad A. Tobagi and Leonard Kleinrock. Packet Switching in Radio Channels: Part II–The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution. *IEEE Transactions on Communications*, 23(12):1417–1433, December 1975. ISSN 0090-6778. doi: 10.1109/TCOM.1975.1092767.

[183] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *3rd International Conference on Embedded Networked Sensor Systems*, SenSys 2005, pages 51–63, New York, NY, USA, 2005. ACM. ISBN 1-59593-054-X. doi: 10.1145/1098918.1098925.

[184] Volker Turau. *Algorithmische Graphentheorie*. Oldenbourg, 3rd edition, 2009. ISBN 978-3-486-59057-9. doi: 10.1524/9783486598520.

[185] Tijs van Dam. *An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks*. Dissertation, Faculteit Electrotechniek, Wiskunde en Informatica, Technische Universiteit Delft, 13. June 2003.

[186] Tijs van Dam and Koen Langendoen. An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks. In *1st International Conference on Embedded Networked Sensor Systems*, SenSys 2003, pages 171–180, New York, NY, USA, November 2003. ACM. ISBN 1-58113-707-9. doi: 10.1145/958491.958512.

[187] Lodewijk F.W. van Hoesel and Paul J.M. Havinga. A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks: Reducing Preamble Transmissions and Transceiver State Switches. In *1st International Workshop on Networked Sensing Systems*, INSS 2004, pages 205–208, Tokio, Japan, June 2004. Society of Instrument and Control Engineers (SICE). URL `http://doc.utwente.nl/64756/`.

[188] Lodewijk F.W. van Hoesel, Stefan O. Dulman, Paul J.M. Havinga, and Harry J. Kip. Design of a low-power testbed for Wireless Sensor Networks and verification. Internal Report TR-CTIT-03-45, CTIT, University of Twente, Enschede, the Netherlands, 2003. URL `http://purl.org/utwente/41407`.

[189] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools 2008, pages 60:1–60:10, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.

[190] *Low Profile Holder Type Crystal Units - XT49S*. Vishay Intertechnology, Inc., March 2010.

[191] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang. EstiNet OpenFlow Network Simulator and Emulator. *IEEE Communications Magazine*, 51(9):110–117, September 2013. ISSN 0163-6804. doi: 10.1109/MCOM.2013.6588659.

[192] Geoffrey Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, and Radhika Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *3rd International Conference on Embedded Networked Sensor Systems*, SenSys 2005, pages 142–153, New York, NY, USA, 2005. ACM. ISBN 1-59593-054-X. doi: 10.1145/1098918.1098934.

[193] Anthony D. Wood and John A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, October 2002. ISSN 0018-9162. doi: 10.1109/MC.2002.1039518.

[194] Chai Wah Wu. *Synchronization in Complex Networks of Nonlinear Dynamical Systems*. World Scientific Publishing, May 2007. ISBN 978-981-270-973-8.

[195] Peng Xie and Jun-Hong Cui. R-MAC: An Energy-Efficient MAC Protocol for Underwater Sensor Networks. In *International Conference on Wireless Algorithms, Systems and Applications*, WASA 2007, pages 187–198, 2007. doi: 10.1109/WASA.2007.37.

[196] Guochang Xu. *GPS – Theory, Algorithms and Applications*. Springer, 2007. ISBN 978-3-540-72714-9. doi: 10.1007/978-3-540-72715-6.

[197] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOMM 2002*, pages 1567–1576, 2002. doi: 10.1109/INFCOM.2002.1019408.

[198] Suyoung Yoon, Chanchai Veerarittiphan, and Mihail L. Sichitiu. Tiny-Sync: Tight Time Synchronization for Wireless Sensor Networks. *ACM Trans. Sen. Netw.*, 3(2):34, June 2007. ISSN 1550-4859. doi: 10.1145/1240226.1240228.

[199] Fan Yu, Tao Wu, and Subir Biswas. Toward In-Band Self-Organization in Energy-Efficient MAC Protocols for Sensor Networks. *IEEE Transactions on Mobile Computing*, 7(2):156–170, February 2008. ISSN 1536-1233. doi: 10.1109/TMC.2007.70719.

[200] Jun Zheng and Abbas Jamalipour, editors. *Wireless Sensor Networks: A Networking Perspective*. John Wiley & Sons, 2009. ISBN 978-0-470-16763-2.

[201] Hubert Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980. ISSN 0090-6778. doi: 10.1109/TCOM.1980.1094702.

List of Figures

2.1 Sensor/actuator node scheme . . . 10
2.2 Radio packet structure . . . 12
2.3 Interesting connected components. . . . 15
2.4 Hidden terminal problem and exposed terminal problem . . . 16
2.5 Communication schemes . . . 17
2.6 Protocol stacks . . . 18
2.7 Synchronous dynamics . . . 21
2.8 State Diagram of a simulated Node . . . 28
2.9 Simulation Process . . . 29
2.10 extDeSIMc Simulator: Start Screen . . . 30
2.11 extDeSIMc Simulator: Simulation Screen . . . 31
2.12 SNoW^5 sensor node . . . 35
2.13 eZ430 Chronos sensor node . . . 36
2.14 SuperG gateway node . . . 37

4.1 Local Clock Implication . . . 50
4.2 Phase neighbor example . . . 52
4.3 Examples for multi-hop conclusions . . . 56
4.4 Snapshots of the progress of desynchronization . . . 60
4.5 Elastic resilience model . . . 62

5.1 Sample scenario of RTS/CTS handshaking . . . 78
5.2 Different communication delays during a packet transmission. . . . 82
5.3 Example of interrupt latency during timestamping. . . . 86
5.4 Propagation of neighbor information – scheme . . . 88
5.5 Propagation of neighbor information – naïve approach . . . 90
5.6 Propagation of neighbor information – phase shift approach . . . 91
5.7 Propagation of neighbor information – reciprocal phase shift approach . . . 93
5.8 Firing packet structure . . . 97
5.9 Frame structure . . . 101

6.1 Phase Shift Diagram . . . 111
6.2 Two-hop Neighbor Information Delay . . . 112
6.3 Topology D_7 "Dumbbell" . . . 113
6.4 Outcome of topology D_7 "Dumbbell" (excerpt) . . . 114
6.5 Impact of ρ on topology D_7 "Dumbbell" (excerpt) . . . 116

7.1 Topology A_{4S} with unidirectional link . . . 127
7.2 Simulation Model Validation (excerpts) . . . 128
7.3 Outcome overlap . . . 129

7.4 Impact of swapped starting times 132
7.5 Randomized start up time result 135
7.6 Topology C_{4S} with sniffer 138
7.7 Outcome of jump size parameter experiment (excerpts) 140
7.8 Topology D_7 "Dumbbell" - reillustrated 142
7.9 Outcome of refractory threshold simulations (excerpts) 143
7.10 Outcome of refractory threshold analysis with $\rho = 0.25$ (excerpt) 144
7.11 Constraint graph $N_C(82)$ 148
7.12 Outcome of simulation of A_{100} (excerpt) 149

List of Tables

3.1 Comparison of PCO with WSN 46

4.1 Survey on MAC protocols implementing primitive of desynchronization . 58

5.1 Survey of different communication delays 82
5.2 Parts of the SmartNET-compatible header 96

7.1 List of simulated swaps in start up order. 133
7.2 Setup parameters when using topology A_{4S} 134
7.3 File size of randomized simulation models 147

List of Listings

2.1 Extract of a sample simulation model 33

5.1 SmartOS function to get node's ID 97
5.2 Calculation of the next time of firing – firing case. 106
5.3 Calculation of the next time of firing – receiving case. 107

6.1 Java's default PRNG implementation 117
6.2 SmartOS-compliant PRNG implementation 118
6.3 Initiating PRNG with node ID as seed 118
6.4 Calculation of the next time of firing with refractory threshold ρ 119

7.1 Dimensioning rules implementation in extDeSIMc 130
7.2 Perl script for start up generation 134

Acronyms

List of abbreviations and the like.

ACK acknowledgment

ADC analog-to-digital converter

C2C car-to-car

C2I car-to-infrastructure

CCA clear channel assessment

CPS Cyber Physical System

CRC Cyclic Redundancy Check

CS Carrier Sense

CSMA Carrier-Sense Multiple Access

CTS clear-to-send

DAC digital-to-analog converter

DCF77 DCF77 time signal

DCM Distributive Cluster Method

DCO Digitally Controlled Oscillator

DS Distribution System

EDS Extended Distribution System

EWMA exponentially weighted moving average

FIFO first in, first out

FTSP Flooding Time-Synchronization Protocol

GPIO General Purpose Input/Output

GPS Global Positioning System

GUI Graphical User Interface

HIL Hardware in the Loop

HTTP	Hypertext Transfer Protocol
I²C	Inter-Integrated Circuit
ID	identifier
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
ISR	Interrupt Service Routine
IRQ	Interrupt Request
IVC	Inter-Vehicular Communication
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LFDM	Local Fault-Tolerant Data Management
LQI	Link Quality Indicator
LUT	lookup table
MAC	Medium Access Control
MANET	Mobile Ad hoc Network
MCU	microcontroller
NED	Network Description
OS	operating system
OSI	Open Systems Interconnection
OLSR	Optimized Link State Routing
P2P	point-to-point
PC	personal computer
PCO	pulse-coupled oscillator
POV	point of view
PRNG	Pseudo Random Number Generator
PSP	phase shift propagation

RAM	random-access memory
RBS	Reference Broadcast Synchronization
RF	Radio Frequency
RNG	Random Number Generator
ROM	read-only memory
RS-232	RS-232 serial port
RSK+	Renesas Starter Kit+
RSSI	Received Signal Strength Indicator
RTS	request-to-send
RTS/CTS	RTS/CTS handshake
RX	Receive, Receive Mode
SBR	Statistic-Based Routing
SPI	Serial Peripheral Interface
SYNC	SYNC word
TC	Topology Control
Tcl	Tool Command Language
TCP	Transmission Control Protocol
TDMA	Time-Division Multiple Access
Tiny-Sync	Tight Time Synchronization
TPSN	Timing-Sync Protocol for Sensor Netowrks
TSS	Time-Stamp Synchronization
TX	Transmit, Transmit Mode
TX/RX	Transmit/Receive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
V2V	vehicle-to-vehicle
VANET	Vehicular Ad hoc Network

WLAN	Wireless Local Area Network
WSAN	Wireless Sensor/Actuator Network
WSN	Wireless Sensor Network
XML	eXtensible Markup Language