TRAFFIC-ADAPTIVE AND LINK-QUALITY-AWARE COMMUNICATION IN WIRELESS SENSOR NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Philipp Hurni

von Fräschels FR

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

TRAFFIC-ADAPTIVE AND LINK-QUALITY-AWARE COMMUNICATION IN WIRELESS SENSOR NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Philipp Hurni

von Fräschels FR

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 22.12.2011

Der Dekan:
Prof. Dr. Silvio Decurtins

Abstract

Research in the field of Wireless Sensor Networks (WSNs) is driven by the severe constraints and the challenges raised by the environments these networks operate in: computational power, memory and energy are scarce and limited resources, and the unreliable nature of the low power wireless channel causes difficulties in all layers of the communication stack. WSNs often operate in *dynamic* environments, where topologies, link qualities and traffic volumes are susceptible to frequent changes at network run-time. In contrast to this, WSN nodes are usually *statically* configured for their intended deployment scenario at compile time, which often leads to suboptimal parameter settings in case the discovered network conditions deviate significantly from the expectations. In practice, it is rather impossible to predict a wide range of network parameters in advance, i.e., the different link qualities inside the network, or the traffic shape and volume that the network will face throughout its entire lifetime. Since the integration of countermeasures for each of the these challenges inevitably causes more strain on the already limited energy budget, *run-time adaptive* mechanisms have recently been proposed, which tune crucial protocol parameters at network run-time and allocate precious resources only *when* and *where* it is necessary.

The main research contributions of this thesis are driven by the research question how to design simple, yet efficient and robust *run-time adaptive* resource allocation schemes within the WSN nodes' communication stack. The thesis addresses several problem domains with contributions on different layers of the WSN communication stack. We first survey the state of the art in MAC protocol design in WSNs with respect to their adaptability to variable traffic. We then introduce our own run-time adaptive MAC protocol, which stepwise allocates the power-hungry radio interface in an *on-demand* manner when the encountered traffic load requires it. We design and evaluate several Forward Error Correction (FEC) strategies to adaptively allocate the correctional power of Error Correcting Codes (ECCs) to cope with timely and spatially variable bit error rates. In the context of TCP-based communications in WSNs, we evaluate distributed caching and local retransmission strategies to overcome the performance degrading effects of packet corruption and transmission failures when transmitting data over multiple hops.

The thesis commences with illustrating two tailormade *frameworks* on which we base our evaluation results. First, our testbed management solution for repeatable experimentation on real-world WSN testbeds is presented, which permits configuration and automated execution of experiments and which served as the main experimental platform for the majority of evaluations of this thesis. Second, we describe our methodology for robust, reliable and accurate *software-based energy-estimation*, which is calculated at network run-time on the sensor node itself.

Contents

Preface

The work presented within this thesis was achieved during my employment as Research Assistant and PhD Student at the Institute of Computer Science and Applied Mathematics (IAM) of the University of Bern. During that time, I was mainly funded over the European-Union research project WISEBED, as well as the *Traffic Adaptive Medium Access Control in Wireless Sensor Networks (TRAWSN)* project financed by the Swiss National Science Foundation (SNF). These two public organizations deserve my honest gratitude for supporting me and giving me the opportunity to pursue my research.

I express my gratitude to Prof. Dr. Torsten Braun, head of the Computer Network and Distributed Systems group (RVS), for supervising this work and for his insightful advises. Prof. Dr. Torsten Braun encouraged and motivated me to publish my results in various reputable conferences and journals. I also thank Prof. Dr. Thiemo Voigt, manager of the Networked Embedded Systems Group of the Swedish Institute of Computer Science (SICS), responsible for the Koreferat of this work, and Prof. Dr. Oscar Nierstrasz, who was willing to be the co-examinator of this thesis.

Many thanks go to my fellow colleagues of the RVS group. Special thanks go to Thomas Staub, Markus Anwander, Zongliang Zhao, Carlos Anastasiades, Marc Brogle and Benjamin Nyffenegger for their good cooperation over the past years. I am especially thankful to the former master students Sebastian Barthlomé and Ulrich Bürgi who diligently contributed to selected implementations and evaluations described in this thesis. Our group secretary Ruth Bestgen further deserves my gratitude for having done an excellent job. Special thanks go to Anton Hergenröder and his colleagues affiliated with ZeuS project from Karlsruhe Institute of Technology for their kind support on the SNMD devices.

Finally, I am deeply grateful to my girlfriend Stefanie Rohrbach for her support and for sharing a wonderful time together. I am grateful to my parents Elisabeth and Ulrich Hurni-Linder, to brother and wife Christoph and Deniz Hurni-Erdemoglu for their support and encouragement. Special thanks for keeping me motivated and having a good time go to my friends Timo Brauchle, Philipp Brändle, Daniel Jenni, Yanick Matter, Milan Nikolic, Pascal Spycher, Lars Urfer and Marco Zimmerli.

Special thanks go to Larry Wall and the developers of the CPAN archive for inventing the *Perl* programming language, which often enough makes an experimental computer scientist's life easier, though sometimes even harder.

Chapter 1

Introduction

The superordinate topic that spans across the different parts of this thesis is driven by the research question how to design simple, efficient and robust run-time adaptive resource allocation schemes within the communication stack of Wireless Sensor Networks (WSNs). This chapter briefly introduces into the WSN field, portrays the fundamental challenges faced in the different parts of the thesis, summarizes the essential contributions and outlines the course of the subsequent chapters.

1.1 Overview

With decreasing cost of integrated circuitry and advanced microdevices, WSNs have emerged as a novel technology for solving various tasks in science and industry where there was no appropriate solution before. WSNs are autonomous networks consisting of a large number of inexpensive small electronic devices and are equipped with sensors to measure a wide range of environmental conditions (e.g., temperature, acceleration, relative humidity, pressure, oxygen concentration).

Figure 1.1 depicts a typical setup of a WSN in an environmental monitoring application. The WSN consists of several distributed nodes and a base station, which is able to communicate with the sensor network over one or several gateway nodes that are connected to the Internet. Nodes are usually deployed across a wide area or around a particular point of interest, in order to monitor, gather and process sensed data of interest, such as different environmental parameters in the forest scenarios depicted in Figure 1.1. Nodes communicate using low power wireless radio interfaces, and collaborate in sensing and forwarding sensed real-world data towards one or more base stations, which may further trigger actions when processing the received data. Since the nodes are equipped with various physical measurement sensors, they can react upon sensed events also by initiating collaborative actions (e.g., a distributed localization mechanism involving several sensors nearby), which is indicated in the Figure 1.1 in the top right corner with a deer being detected or in the bottom with a fire being detected. A good introduction into the operation characteristics of sensor networks, algorithmic issues of distributed sensing systems, their potential applications and limitations can be found in *Aky-*

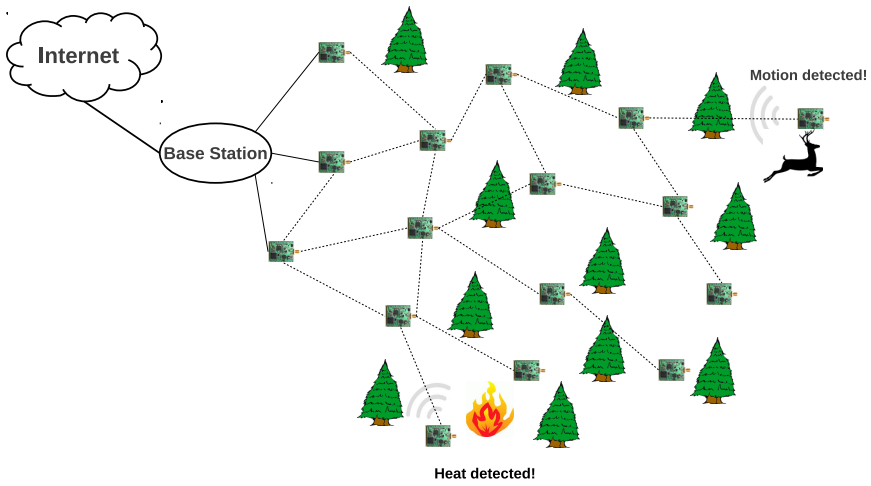


Figure 1.1: Wireless Sensor Network Application Scenario

ildiz et al. [3] and *Pottie et al.* [149]. The survey studies [4][7] further discuss properties and novel challenges of the emerging field of wireless multimedia sensor networks (WMSNs).

For decades, measurements of physical values were carried out manually with bulky and costly equipment, a tedious, time-consuming and error-prone procedure. The convenient and automated manner of being able to gain real-time measurement data from all kinds of distant locations and in a high resolution is the key advantage that drives research and innovation on WSN technologies today. Mark Weiser's prominent Vision of the *Computer of the 21st Century* [181] or the visionary *Smart Dust* paper [100], which both describe a future where computing resources are embedded ubiquitously into our environment yet only give an idea where the journey leads. WSNs are more and more applied in different domains of our everyday life. Applications have become numerous and cover a wide range of problems. They contain but are not limited to the following categories:

- **Environmental Monitoring in the Natural Sciences:** WSNs have been applied to monitor various problems and phenomena in the natural sciences. A well-known example is the study on the nesting and breeding behavior of storm petrels conducted by *Mainwaring et al.* [118]. *Pasztor et al.* [138] and *Dyo et al.* [54] examine the social patterns of European badgers during one year deployment of an automated wildlife monitoring system in a forest environment, using a mobile WSN attached to the badgers. *Barrenetxea et al.* [16] and *Beutel et al.* [19] collect long-term high-altitude alpine temperature and other climate-related data. *Werner-Allen et al.* [183] use a WSN to monitor the eruptions of active and hazardous volcanoes.
- **Event Detection and Tracking Applications:** WSNs have been used for specific use cases related to tracking people or objects and localizing the event of interest. Prominent examples of this class of applications is the sniper lo-

calization use case PinPtr [165], which uses a WSN to detect muzzle blasts and acoustic shockwaves and which, based on measuring the time of arrival of the acoustic waves on different locations of the network, can localize the ignition. *Wittenburg et al.* develop a collaborative event detection system built to monitor fenced areas and prevent intrusion in [186]. A permanent 24/7 solar-operated network for fire detection and localization has been recently installed in the Asturias region in the north of Spain [98]. The covered area is about 210 hectares large and several tens of nodes are permanently attached to trees to continuously measure temperature and humidity conditions.

- **Disaster Aid and First Response Systems:** *Gao et al.* [67] develop a system facilitating collaborative patient care in emergency response and mass incident scenarios. The system relieves the workload for each first response collaborator, and delivers pertinent information to first responders in order to effectively treat a large number of patients within a short time.
- **Logistics, Business and Industrial Process Control:** Logistics and supply-chain management are more and more taking advantage of market-ready solutions to visualize and keep track of warehouses and supply chains. *Evers et al.* [61] study potential applications in this domain, where a simple and ready-to use WSN platform *SensorScheme* has been proposed. Numerous companies have evolved (e.g., Crossbow [37], Dust Networks [53], Micro Strain [125], Ambient Systems [8] or SowNet Technologies [167] only to name a few examples) which are integrating WSN technologies into the manufacturing, warehouse or supply-chain management processes in various industries.
- **Health-care related Systems:** WSNs are more and more applied in the context of health-care. With the *CodeBlue* [119] architecture, the authors have explored the applicability of WSN technology in a hospital environment to monitor the vital signs (pulse, electrocardiogram data) of patients. *Chipara et al.* [35] conduct a long-term study of a wireless clinical monitoring system collecting pulse and oxygen saturation readings from patients. The system utilized in the study reached an end-to-end event reporting reliability of more than 99% and included 41 patients supervised over 7 months.

The above list is just an excerpt from many application domains which have been proposed and sometimes developed into market-ready products in the past couple of years. WSN technologies are furthermore increasingly applied in problem domains related to military, police and border protection, for precision agriculture, intelligent buildings (smart home environments) or civil engineering and structural health monitoring applications (e.g., dam, bridge or tunnel surveillance).

Although the application domains may vary heavily, the challenges and the hard restrictions the employed WSNs are subject to, are similar in each many of the use cases. The subsequent Sections 1.2 and 1.3 briefly discuss the main challenges that are typically faced in research and development of WSN systems and technologies, and localizes the main contributions of this thesis in the context of the typical WSN communication software stack.

1.2 Problem Statement

In this thesis, we contribute to repeatable methodologies for experiment-driven empirical research in the WSN field (Part I) and address several inherent challenges in the context of WSN communication protocols (Part II). Our contributions tackle issues relating to the following problem domains:

Experiment Methodologies: Experimental evaluations and verifications of WSN mechanisms and protocols have since long been carried out with basically two paradigms: *network simulation* or *real-world prototyping* followed by *testbed-based evaluation*. With research in the WSN field growing more mature, the latter has become increasingly important: researchers more and more aim at investigating the real-world feasibility of their proposed protocols and mechanisms on real-world devices in experimental testbeds. The usual approach consists in rapid-prototyping a WSN algorithm or application first by using a network simulator, testing and comparing it to existing approaches, and proceeding with implementing a real-world prototype when the simulation results are promising. The efficient and convenient operation of WSN testbeds and experimentation with the latter requires management functionalities, for which off-the-shelf solutions are yet missing. The representation of experimental data is another domain where different formats severely aggravate experimental research. Experiment results from large-scale experimental WSN studies, which could be helpful for many other researchers working in the same field, are often arbitrarily organized and formatted. We contribute to this problem domain with our management software framework for repeatable experimentation in WSNs testbeds (cf. Section 1.3.1), that integrates a standardized notation of experiment results.

For years, experimental research in the field of energy-aware and energy-conserving protocols in distributed systems has required weeks or even months of tedious and time-consuming use of bulky cathode-ray oscilloscopes or high-resolution multimeters for real-world evaluation. The recent establishment of *software-based* energy-estimation techniques has been a significant contribution, yielding a simple and painless, but yet accurate methodology for energy estimation on the running systems themselves. The availability of such software-based estimations is furthermore a cornerstone towards real-world applicability of many proposed energy-aware protocols of the last decade, e.g., in the domain of energy-aware routing or clustering algorithms, since such mechanisms often rely on having an indication about the consumed and the residual energy resources on the nodes themselves at run-time. We contribute to the *proliferation* of software-based energy estimation by an in-depth analysis of their estimation accuracy (cf. Section 1.3.2).

Hardware Limitations: Several factors determine the technical restrictions and physical constraints of wireless sensor nodes. One major decisive factor is the economic costs per unit. The costs per unit has to remain low (in the magnitude of less than 100 USD per piece) to make a wide range of applications in business and industry possible. A sensor node is therefore usually a small sized device of

a few square centimeters equipped with inexpensive low-power electronic components, e.g., a 16-bit microcontroller with some 60 KBytes of ROM and 10 KBytes or even less RAM, a cheap low-power radio chip and various inexpensive digital sensors. To the best of our knowledge, however, there is yet no commercially available sensor node platform that reaches the often cited visionary per-unit costs of "less than one US dollar". The restrictions with respect to memory and processing power pose inherent challenges to the development of sensor network software, ranging from the application layer all the way down to the operating system and the communication stack. Often enough, WSN mechanisms designed and evaluated in simulators have to be *downsized* and *simplified* in order to realize them on real-world sensor network platforms. The restrictions in memory and computational power have been encountered throughout all the contributions presented in this thesis. Whether the contribution was targeting at MAC-layer or transport layer issues, we have addressed these restrictions generally by keeping our systems *as functional as necessary* but as *simple as possible*.

Limited Energy Resources: Sensor nodes should consume as little energy as possible, since real-world sensor networks should remain operable in an area of interest over several days, weeks or even years, given an initial battery charge, e.g., of two AA batteries. However, when keeping the radio and the onboard sensors permanently active, a typical sensor node drains out of energy after not much more than a couple of days. In the past decade, mechanisms have been developed to prolong the time a node can live on an initial energy charge, especially by designing energy-conserving communication protocols on the MAC and routing layer. We contribute to this problem domain with our contributions on flexible energy-efficient medium access control, which eliminates the major portion of energy-waste, yet taking Quality of Service goals into considerations.

Quality of Service Requirements: For a long time, the major research goal in the MAC area consisted in *minimizing* the energy consumed by the sensor nodes at any cost, even though essential service characteristics had to be sacrificed. In the past couple of years, however, applications requiring higher Quality of Service and low latency during certain intervals of intense activity have emerged, e.g., wireless multimedia sensor networks (WMSNs) transmitting a large image once in a while or healthcare-related systems registering anomalies in their sensed values. In order to meet the requirements of such applications, run-time adaptive mechanisms have emerged across all layers of the communication stack. These adaptive mechanisms generally attempt to find a fair balance between the design goals of energy-conservation, energy-efficiency and the offered Quality of Service. We contribute to this activity with our run-time adaptive MAC protocol, which bases on established design principles of a decade of research on energy-efficient medium access control, and which is capable of accommodating to variable traffic based on a customizable and user-defined set of rules and parameters (cf. Section 1.3.3).

Unreliable Wireless Channel: A key factor for the proliferation of WSN technologies is the reliability of the communication: it is crucial for many applica-

tions that the sensed data is delivered quickly and reliably across the network. The low-power wireless channel used in sensor networks, however, is prone to a wide range of wireless phenomena, which may ultimately result in packet corruption and packet loss, such as high bit error rates due to multipath propagation, reflection and scattering effects, interferences with nearby nodes or other devices. Tackling these challenges requires sophisticated mechanisms integrated into the communication stack, ranging from the medium access control layer, over error-resilient routing and data dissemination layers, to intelligent mechanisms integrated into the transport layer. All components further have to take the inherent challenges of unreliable and lossy links, but also the energy restriction and memory/computational limitations into consideration. In this thesis, we tackle problems related to lossy and unreliable links in two different contexts. First, by providing countermeasures to bit errors on the link layer by applying sophisticated Forward Error Correction schemes (cf. Section 1.3.4), and second, by means of a distributed caching and local TCP regeneration mechanism operating across multiple hops in the context of TCP/IP in WSNs with radio duty-cycled MAC protocols. (cf. Section 1.3.5).

1.3 Contributions

Part I of this thesis tackles the problem space of efficient and repeatable experimentation methodologies. We inquire into issues related to WSN testbed design and management, as well as accurate and robust software-based energy-estimation. In Part II, we address the challenges related to communication in WSNs outlined in Section 1.2, which are raised by the severe restrictions and the adverse circumstances of WSN environments. We introduce novel mechanisms in different layers of the communication stack. Figure 1.2 schematically depicts the protocol stacks of three nodes involved in a communication process: a source node, an intermediate node and a destination node, and localizes the contributions of this thesis.

- *Wireless Sensor Network Testbed Design and Management:* We contribute to testbed-based and experiment-driven research methodologies with our reusable management architecture, which permits repeatable and automated experimentation and makes the testbeds fully available over the Internet (cf. Section 1.3.1).
- *Accurate and Robust Software-based Energy-Estimation:* We contribute to the proliferation of software-based energy-estimation methodologies, which have become increasingly important in the field of energy-conserving protocol development for wireless sensor and ad-hoc networks (cf. Section 1.3.2).
- *Adaptive Medium Access Control:* We thoroughly analyze the state of the art in Energy-Efficient MAC (E^2-MAC) protocols. We introduce a run-time adaptive MAC protocol, which allocates the power-hungry radio interface in an *on-demand* manner when the traffic level requires it (cf. Section 1.3.3).
- *Link-Quality-Aware Adaptive Forward Error Correction:* We tackle the problem of bit errors caused by the error-prone low-power wireless channel by ap-

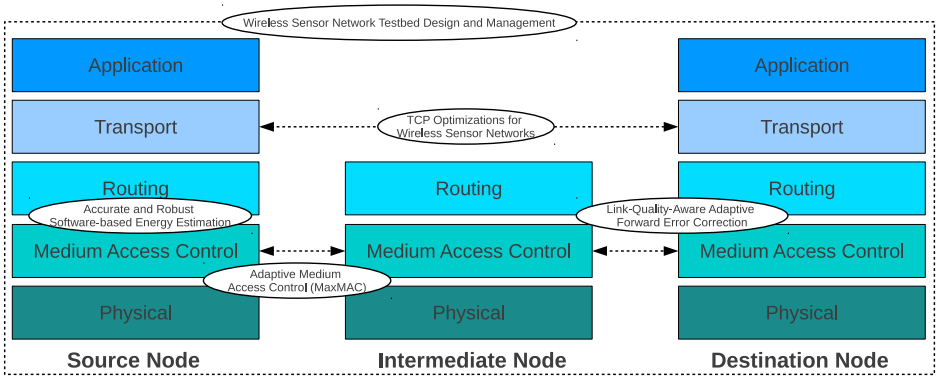


Figure 1.2: Overview of Thesis Contributions

plying Forward Error Correction (FEC) mechanisms in an *on-demand* manner *when* and *where* the channel circumstances require it (cf. Section 1.3.4).

- *TCP Optimizations for Wireless Sensor Networks:* We tackle the problems raised by random link-layer packet loss due to the unreliable error-prone wireless channel, which causes significant strain on the energy resources and severely deteriorates the end-to-end throughput and latency. (cf. Section 1.3.5).

1.3.1 Wireless Sensor Network Testbed Design and Management

In the past couple of years, numerous universities and research institutions have set up real-world sensor network testbeds for teaching and research, i.e., the evaluation of real-world behavior of developed protocol mechanisms. An increasing number of stationary WSN testbeds have been put into operation, with different node hardware and sometimes significantly differing architectural testbed design. The most prominent examples are Harvard University's MoteLab, the TWIST testbed of TU Berlin, or the Kansei testbed of Ohio State University. The management solutions implemented for these testbeds have, however, often been tightly coupled to one particular testbed deployment, and are hence not easily reusable for further testbed setups. Still today, researchers setting up an own testbed are often starting from scratch to implement testbed management features, which are as simple as user account management, experiment resource reservation, configuration and scheduling, or a consistent representation of experiment results.

We bridge this missing gap with our Testbed Management Architecture for WSN Testbeds (TARWIS), which we describe in detail in Chapter 3. TARWIS is a generic and reusable management framework for repeatable experimentation in testbeds of WSNs, which has been kept independent from the underlying testbed organization, the sensor node hardware and software. Our practical experience with several researchers has shown that TARWIS is a powerful instrument for controlled and repeatable experimentation in WSNs. We have conducted the major part of experiments described in this thesis using TARWIS, among them most of the results of Chapters 7, 8 and 9.

1.3.2 Accurate and Robust Software-based Energy-Estimation

In order to examine real-world WSN communication protocols with respect to their energy consumption, a simple, repeatable and reasonably accurate methodology to assess the energy consumption of a sensor node is an important cornerstone, since physical measurements using oscilloscopes or high-resolution multimeters are time-consuming and costly. In the past couple of years, software-based energy-estimation has become a widespread technique to quantify the energy consumption in WSNs. Many prominent E^2-MAC protocol studies have entirely relied their experimental results upon software-based estimations. More and more research papers employ this methodology, although no existing study has yet thoroughly validated the accuracy of this approach with physical hardware-based energy measurements. Software-based energy estimation techniques clearly have their advantages and drawbacks: software-based estimation can only deliver *estimates*, and introduces inherent side-effects, as the estimation mechanism itself causes computational costs, which are hard to account for. The advantages, however, are manifold: with an energy estimation being present on the node at run-time, numerous energy-aware algorithms can be put into practice in real-world deployments.

We contribute to the research field of software-based energy estimation by systematically studying their *estimation accuracy* on one of our employed sensor hardware platforms. We have evaluated several energy estimation models with prototype implementations of four different wireless channel protocols, and have run a large number of experiments. By varying the key parameters (e.g., traffic rate, node instance) over a wide range, we were able to statistically describe their estimation accuracy. Our proposed methodology requires careful calibration with prior hardware-based energy measurement tools, which we describe in Section 2.3. The software-based energy-estimation methodology and our evaluation results with respect to the accuracies are then presented in Chapter 4.

1.3.3 Traffic-Adaptive Medium Access Control

Our first contribution related to traffic-adaptive MAC design is formed by the evaluation of the WiseMAC burst transfer mode. We propose an enhanced scheme, which addresses the problem of tree-based scenarios where many nodes transmit packets to one bottleneck node. In order to cope with higher traffic, we propose to enhance the original scheme allowing bottleneck nodes to temporally abandon their sleep-wake pattern. The enhanced scheme is shown in Chapter 5 in simulation and by means of a prototype to increase throughput by roughly 20%.

Many of today's Energy-Efficient Medium Access (E^2-MAC) protocols have been designed with the only goal of minimizing the energy conservation. As a result, most of these protocols are able to deliver little amounts of data with a low energy footprint, however, introducing severe restrictions with respect to the achievable throughput and latency, and totally failing to adapt to varying traffic loads and changing requirements of the imposed traffic load. The gain in energy-efficiency

hence comes at the cost of severely restrained maximum throughput, as well as massively increased end-to-end packet latency. Chapter 6 thoroughly examines the design space of the six most frequently cited E^2-MAC protocols and examines their performance under variable traffic load. We then develop a tri-partite metric to measure and quantify the *traffic adaptivity*. The metric takes into account the crucial target variables maximum achievable throughput, latency and energy-efficiency, and maps the protocol's performance to a one-dimensional number.

With the introduction of the *Maximally Traffic-Adaptive MAC (MaxMAC)* protocol in Chapter 7, we specifically target at alleviating the performance degrading impact discovered in Chapter 6 of most of today's E^2-MAC protocols with respect to traffic adaptivity. Relying on best practices of a decade of E^2-MAC protocol research, we base our investigations on adaptable protocol mechanisms on the most widely applied class of asynchronous random-access and preamble-sampling based MAC protocols. While MaxMAC operates with a low energy footprint at low traffic, it is able to dynamically adapt to sudden changes in the network traffic load at run-time. It integrates established design principles of asynchronous preamble-sampling based MAC protocols with novel run-time traffic adaptation techniques to allocate the costly radio transceiver truly in an *on demand* manner.

Chapter 7 compares MaxMAC against a selection of existing E^2-MAC protocols. By applying the metric defined in Chapter 6 to network simulation results of MaxMAC, we show that the developed protocol mechanisms succeed in reaching a high *traffic adaptivity*. The chapter then continues with thoroughly evaluating our MaxMAC prototype implementation and comparing it against other wireless MAC protocols. The evaluation is conducted using the TARWIS management architecture presented in Chapter 3, using our distributed testbed facilities.

1.3.4 Link-Quality-Aware Adaptive Forward Error Correction

The low-power wireless channel of WSNs is prone to a wide range of wireless phenomena. High bit error rates are caused by multipath propagation, reflection and scattering effects, interferences with nearby nodes or other electronic devices using the same or a nearby band. Forward Error Correction (FEC) mechanisms are able to correct a certain number of errors without making a retransmission necessary. FEC is based on Error Correcting Codes (ECCs), which encode data in such a way that a certain number of random bit flips due to transmission errors can be recovered. Applying FEC in WSNs yields the crucial design question to select an appropriate ECC for a given network and application. While a too weak code might not be able to correct many errors, a too strong code would constitute a waste of time and energy spent for encoding and decoding. In Chapter 8, we therefore propose to *adaptively* select among different ECCs at *run-time* rather than compile-time, and to choose different codes for each individual link based on the individual link reliability, instead of applying network-wide settings.

In Chapter 8 we explore the potential of FEC schemes in general, and run-time

adaptive ECC selection schemes in particular, in the context of WSNs with links of different transmission success rates and error occurrence patterns. We have implemented eight different ECC codes in our library *libECC* and have proposed three run-time adaptive Forward Error Correction strategies. The implemented schemes were examined in several real-world experiments, including indoor and outdoor links and taking into account single-hop and multi-hop topologies. The evaluation is conducted using the TARWIS management architecture presented in Chapter 3 with our distributed testbed facilities.

1.3.5 TCP Optimizations for Wireless Sensor Networks

Wireless sensor nodes and other small embedded devices become increasingly accessible over the Internet, typically using one or more gateway nodes which negotiate between the WSN and the IP-based Internet. The development and the proliferation of the well-known µIP stack has made TCP/IP-based communication possible within WSNs: single nodes can nowadays be accessed using common TCP/IP-based tools and applications, as for example Telnet, SMTP or FTP. TCP/IP has been shown to perform rather poorly in WSNs with multiple hops, due to the unreliable nature of the wireless channel (higher bit error rates and packet losses), particular properties of and interactions with the underlying wireless MAC protocols (exponential backoff mechanisms, hidden node and exposed node problem), and the design of the TCP congestion control mechanisms.

In Chapter 9, we tackle these deteriorating effects on the end-to-end TCP throughput by introducing recently proposed distributed caching and local retransmission mechanisms, as well as self-developed extensions of the latter. We introduce our MAC- and application-layer independent *Caching and Congestion Control (cctrl)* module, which situates just below the µIP stack. We show in a series of experiments that *cctrl* is able to significantly increase the TCP throughput across multiple hops when operating with various MAC protocols, among them three radio duty-cycled E^2-MAC protocols. The evaluation is conducted using the TARWIS management architecture presented in Chapter 3 with our distributed testbed facilities.

1.3.6 Summary of Contributions

The main contributions of this thesis can be summarized as follows:

- We designed and implemented the generic and reusable *Testbed Management Architecture for Wireless Sensor Networks* TARWIS. This testbed management framework to date manages a testbed of 47 wireless sensor nodes located in the two buildings of the Institute of Computer Science and Applied Mathematics of University of Bern on the Engehalde Campus, making it available over the Internet for other researchers. Besides our own testbed, TARWIS has been deployed in eight other testbeds all across Europe, with node deployments between few tens of nodes to over 100 nodes.

- We systematically studied the estimation accuracy and limitations of software-based energy estimation methodologies with various wireless MAC protocols and wide range of traffic intensities. We proposed a model enhancement to today's most frequently used software-based model, and a methodology to calibrate the former, which is based on calculating the crucial model parameters with statistical methods using empirical measurement data.
- We evaluated the performance of the WiseMAC burst transfer mode *More Bit* and proposed an enhanced scheme, which addresses the problem of bottleneck nodes in tree-based scenarios. The enhanced scheme is shown in simulation and by means of a prototype to increase throughput by roughly 20%.
- We examined the behavior of a selection of the most well-known Energy-Efficient MAC (E^2-MAC) protocols under variable traffic conditions in a network simulator environment. We introduced a formal notion to assess and quantify the *traffic adaptivity* of wireless sensor MAC protocols using a tripartite metric, which takes into account the crucial variables of latency, energy-efficiency and the maximum achievable throughput.
- We proposed the Maximally Traffic-Adaptive MAC (MaxMAC) protocol, which provides support for scenarios with timely variable traffic conditions. We have shown in simulation and by means of a real-world prototype implementation, evaluated on our testbed facilities, that the protocol reaches the throughput and latency of energy-unconstrained CSMA in situations of high traffic, yet exhibiting a high energy-efficiency under sparse low-rate traffic.
- We studied the potential of (adaptive) Forward Error Correction in WSNs. We further developed three run-time adaptive strategies to select suitable Error Correcting Codes to efficiently tackle and alleviate the performance degrading impact of spatially and temporally variable bit error patterns at network run-time.
- We evaluated the impact of distributed caching and local retransmission strategies on the end-to-end throughput in TCP/IP-based WSNs. We have integrated our proposed techniques into a MAC- and application-layer independent module below the µIP stack in the Contiki OS, and could significantly increase the end-to-end TCP throughput in numerous experiments.

1.4 Thesis Outline

Chapter 2 introduces and discusses the most important and most significant related work we rely upon in our main contributions.

Chapter 3 portrays our management framework TARWIS we used for the major part of the experiments, and the campus-wide 47-nodes WSN testbed it manages.

Chapter 4 discusses our contributions towards robust and accurate software-based energy estimation mechanisms running on the nodes themselves.

Chapter 5 describes and evaluates our extension of the WiseMAC burst transfer

mode in a network simulator and with a real-world prototype implementation.

Chapter 6 surveys the current state of the art of Energy-Efficient (E^2-MAC) MAC protocols with respect to their run-time traffic adaptability in a network simulation environment, and motivates our subsequent contributions in this field.

In Chapter 7, we propose the novel Maximally Traffic-Adaptive MAC (MaxMAC) protocol, which is an E^2-MAC protocol aiming at run-time traffic adaptability. We evaluate the protocol in simulation against a selection of E^2-MAC protocols and present a proof-of-concept implementation of MaxMAC on real-world devices. We evaluate this real world prototype against other wireless MAC protocols and thoroughly evaluate its advantages and drawbacks.

In Chapter 8, we explore the potential of FEC techniques in general, and adaptive strategies in particular. After examining eight different ECCs applied for each link in a static manner, the chapter studies three different run-time adaptive FEC techniques, which vary the current ECC based on the current link quality.

Chapter 9 discusses the performance degrading impact of link-layer packet losses on TCP/IP-based communication in WSNs. The chapter evaluates our proposed countermeasures to improve TCP throughput in multi-hop WSN topologies.

Chapter 10 concludes the thesis, discusses the future outlook of ongoing work on the discussed problem domains and motivates potential topics for future research.

Chapter 2

Related Work

This chapter discusses the most important and most significant related work we rely upon for our contributions presented in the subsequent chapters of the Parts I and II of this thesis. The order in the discussion of the related topics in this chapter corresponds to the order of the chapters that present our main contributions.

The chapter commences with Section 2.1 portraying the evaluation platforms and WSN operating systems that were used for several real-world measurements and evaluations. Section 2.2 then discusses architectural properties and issues related to network and experiment management of the most well-known testbeds of WSNs. Then, the chapter presents a collection of methodologies and tools to measure the energy consumption of WSN nodes in Section 2.3. In order to evaluate our contributions in Part II of the thesis, several methodologies for *hardware-based* energy-measurement, and *software-based* energy-estimation have been employed. Section 2.3 discusses both approaches, the physical hardware-based methodologies basing on dedicated measurement hardware in 2.3.1 and 2.3.2, and recent literature about software-based energy-estimation techniques in 2.3.3.
The chapter then discusses the research field of Energy-Efficient MAC (E^2-MAC) protocols for WSNs in Section 2.4 by categorizing the most well-known protocols with their advantages and drawbacks, relating them to our contributions and giving an outlook about future trends and developments. The chapter continues with outlining the fundamentals of a selection of Error Correcting Codes (ECCs) in Section 2.5, on which we build upon for designing adaptive Forward Error Correction (FEC) strategies. Section 2.6 discusses literature related to TCP/IP over multiple wireless hops in wireless networks in general, and in sensor networks in particular.

2.1 Evaluation Platforms

In this section we briefly portray the sensor hardware platforms and sensor node operating systems used for the various prototype implementations throughout this thesis. The *Embedded Sensor Board* and the first version of the *ScatterWeb Operating System* presented in Sections 2.1.1 and 2.1.2 were only used for our preliminary study on extensions of the E^2-MAC protocol WiseMAC in Chapter 5. The succes-

sor of this platform, the *Modular Sensor Boards (MSB)* along with the *ScatterWeb*^{2} *Operating System* are presented in Sections 2.1.3 and 2.1.4. The MSBs been used for the major portion of experiments and evaluations of this thesis, in particular in the Chapters 4, 7 and 8. Finally, Sections 2.1.5 and 2.1.6 discuss the prominent *TmoteSky/TelosB* platform and the *Contiki Operating System*, which we used for our contributions on TCP across radio duty-cycled multi-hop WSNs in Chapter 9.

2.1.1 Embedded Sensor Board

The Embedded Sensor Boards (ESB) [158] (cf. Figure 2.1) belong to the first sensor hardware platforms of the European research community on WSNs. The ESBs have been developed at Freie Universität Berlin and the spin-off company ScatterWeb GmbH [157]. Each ESB is equipped with a MSP430 microcontroller, a TR1001 [156] radio transceiver, 32k EEPROM, RS232 port, a JTAG port to flash and debug the CPU, an audible beeper, and a number of sensors (passive infrared, vibration/tilt, microphone, temperature). The node is powered using three AA batteries on the battery rack mounted behind the board.

Figure 2.1: Embedded Sensor Board (ESB)

2.1.2 ScatterWeb Sensor Operating System

The ScatterWeb (v.1) Operating System is a very small and simple sensor operating system. This event-driven, single-threaded sensor node OS is entirely written in C, is well documented and the source is open to a large extent. The control flow of the ScatterWeb OS core is a simple never-ending loop function named *superloop* that calls handler-functions if and event has to be processed (e.g., for timers, for handling serial IO or radio transmissions) and lets the CPU enter the sleep mode LPM1 for a certain time if not, until the next iteration of the superloop.

The superloop contains a function that resets the timer register of the watchdog. If the program gets stuck in a loop during the following statements of the main-loop, the watchdog timer runs out and causes the system to be reset. When all pending handler-functions of one superloop iteration have been called, the watchdog is stopped and the microcontroller enters the low-power mode LPM1 again.

2.1.3 Modular Sensor Board

The Modular Sensor Board (MSB430) node is the successor of the ESB node and was also developed by Freie Universität Berlin and ScatterWeb GmbH [157]. This modular node platform features the following main components on its *core module*:

- A Texas Instruments MSP430 series RISC CPU (MSP430F1612): this CPU can be clocked with 100 kHz up to 11 MHz. The clock speed can be adapted by a software configurable digital controlled oscillator (DCO). The MSP430F1612 has 55 KB flash memory and 5 KB RAM, and further 18-digital I/O pins connected to analog-to-digital (ADC) and digital-to-analog (DCA) converters.
- A CC1020 Chipcon [173] configurable wireless radio transceiver using a low-noise amplifier that operates in the ISM-band around 868 MHz. Its output power reaches an amplitude up to 8.6 dBm (7.2 mW). The CC1020 uses 8 channels with a data rate of 19.2 kbit/s when using Manchester encoding. The CC1020 would technically support a raw transmission rate of up to 153.6 kbit/s, however, this modulation is currently not supported by ScatterWeb^2 OS.
- A Secure Digital Memory Card (SD) Reader can store large amounts of data on Secure Digital High-Capacity (SDHC) cards with a capacity up to 32 GB.
- A Temperature and Humidity Sensor Sensirion SHT11 [161] is capable of measuring temperature and relative humidity.
- A Freescale MMA7260Q Accelerometer [66] is capable of measuring the acceleration in 3 dimensions (x,y,z).

The MSB430 *core module* makes most of the digital IO ports accessible for the extensions. Hence, on top of the core module, one or more board extensions can be plugged in, as illustrated in Figure 2.2. The MSB430 sensor node can hence be conveniently *customized* by the end user for certain special tasks, plugging in additional modules, which renders a complete redesign of the board obsolete.

Figure 2.2: MSB430 Sensor Node Platform with pluggable Sensor Modules

2.1.4 ScatterWeb^2 Sensor Operating System

The MSB430 sensor boards can be run with the ScatterWeb^2 OS. This rather simple OS was developed at Freie Universität Berlin and ScatterWeb GmbH. ScatterWeb^2 can be seen as the successor of the ScatterWeb OS (v.1) for the ESB nodes described in Section 2.1.2. However, since there are no well-defined interfaces between network subsystem and MAC, any implementations related to MAC protocol mechanisms turned out not to be portable to the newer MSB430 node.

The event-driven, single-threaded ScatterWeb^2 OS is entirely written in C and integrates drivers for the CC1020 radio module, the onboard sensors and SD card reader. The core of the OS consists again in the so-called *superloop* which is preempted by interrupts. Interrupts are triggered by the components on the board, e.g., the radio chip, the sensors, or the UART serial interface. *Handler functions* called in the *superloop* then process the events (e.g, Net_rxHandler, Timers_handler).

The ScatterWeb^2 OS provides an intuitive user interface and a simple means to add and define commands, with which the user can communicate with the sensor node over the serial interface. The network stack is simple and by default consists in a IEEE 802.11-like random backoff CSMA (without RTS/CTS) on the MAC layer, and offers functions for acknowledged or unacknowledged unicast and broadcast.

The popular sensor node operating system Contiki has a port for the MSB430 sensor boards since Contiki v.2.4.

2.1.5 TMote-Sky/TelosB Node

The TmoteSky/TelosB platform depicted in Figure 2.3 is one of the most popular and most frequently used sensor node platforms. It was designed for low power and high data-rate sensor network applications. It features the following components:

- A Texas Instruments MSP430 series RISC CPU (MSP430F1611) offering some 48kB of ROM and 10kB of RAM.
- A Chipcon CC2420 [174] radio transceiver, which is an IEEE 802.15.4 compliant radio for wireless communications operating in the 2.4GHz ISM band. The radio provides a faster data rate (250 kbps) compared to the MSB430 boards, however, at a price of a lower range.
- The onboard temperature and humidity sensor Sensirion SHT11 [161] is capable of measuring temperature and relative humidity.
- An additional Numoniyx Forte M25P80 Flash memory [135] provides 1024kB of external space, e.g., for code or for logging data.

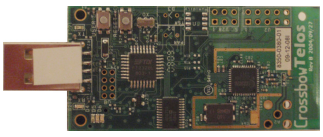


Figure 2.3: TmoteSky/TelosB Sensor Node Platform

2.1.6 Contiki Operating System

The Contiki Operating System [47] is an open source operating system designed for networked embedded systems with small amounts of memory, supporting a wide range of target platforms. Contiki to date supports various microcontroller chips, ranging from 8-bit over 16-bit to 32-bit architectures. A typical Contiki configuration's size is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki features an event-driven kernel, providing support for pre-emptive multi-threading using *protothreads* [51]. Protothreads are lightweight threads that provide a linear, thread-like programming style on top of Contiki's event-driven kernel. Another key feature of Contiki is its support for dynamic linking of code at run-time. This facilitates over-the-air programming and integrating new functionalities without the need of collecting and reprogramming the nodes offline.

A core component of Contiki is its modular and highly customizable network stack, which has well-defined generic interfaces for various node platforms. In contrast to, e.g., the ScatterWeb^2 OS, which is limited to the MSB430 nodes and some further developed successors of this node type, most components Contiki's network stack can be run on all ports supported by Contiki. The structure of this stack is illustrated in Figure 2.4. The lowest component of the Contiki network stack is the radio driver, which is platform-dependent, but which offers standardized services for the layers above. On top of this lies the MAC layer, followed by the Rime layer.

Contiki Rime

Contiki Rime [49] is a collection of different node-to-node communication services. These services are implemented in a hierarchical manner, with lower layer services providing basic features, and more complex services building on top of them. Figure 2.4 depicts the Rime services and their underlying hierarchy. Services or applications on layers above Rime can invoke any Rime service. The connection is usually established by copying the data to the so-called *Rime buffer* and initiating the actual transmission. The Rime protocols that are supported to date are the following:

- **abc:** Anonymous Best-effort Single-hop Broadcast protocol
- **ibc:** Identified Best-effort Single-hop Broadcast
- **uc:** Best-effort Single-hop Unicast protocol
- **stuc:** Stubborn Single-hop Unicast protocol
- **ruc:** Reliable Single-hop Unicast protocol
- **mh:** Best-effort Multi-hop Unicast
- **rmh:** Hop-by-hop Reliable Multi-hop Unicast
- **polite:** Polite (Anonymous) Single-hop Broadcast protocol
- **ipolite:** Polite (Identified) Single-hop Broadcast protocol
- **nf:** Best-effort Network Flooding

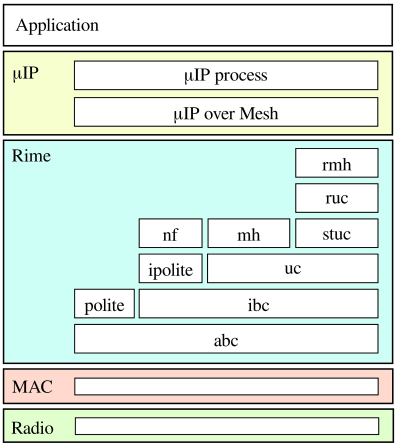


Figure 2.4: The Contiki Network Stack

The service names basically describe the properties and features of these services. More elaborate and detailed information can be found on the Contiki OS [47] website and discussion board, which has a steeply increasing number of subscribers and a remarkably active user and developer community.

Contiki µIP Stack

Contiki's µIP Stack was first introduced by *Dunkels et al.* [45] in 2003, hence even before the development of Contiki itself. The small µIP Stack is a minimalistic implementation of the most important functionalities of the TCP/IP suite. Starting from its initial support of IPv4, it has been augmented in the recent past to support IPv6, as well as UDP and ICMP. To date, µIP implements the basic requirements for hosts in the Internet as specified by RFC1122 [23]. Contiki's µIP Stack can be expected to seamlessly operate with almost any platform providing a RFC-compliant TCP/IP stack, in fact it is already used in various embedded devices in the industry today.

In a trade-off between code size and and RFC-compliance, some features of the functionalities defined in RFC1122 had to be left out, e.g., the Address Resolution Protocol (ARP) or the limitation of the µIP stack to not support the transmission of multiple unacknowledged TCP segments, hence keeping a limitation of a congestion window set to the value of 1 forever. This of course results in a limited maximum achievable throughput, but prevents nodes from allocating precious memory for unacknowledged segments.

Contiki MAC Layer Protocols

Thanks to the large user community of the Contiki OS, a notable collection of protocols and applications have evolved in the past years, which often work on several of the Contiki-supported target platforms. In the course of this thesis, we made use of Contiki's selection of readily available MAC protocols of Contiki

v.2.4 and Contiki v.2.5-rc (*release candidate* as of May 2011). We briefly discuss the protocol implementations we used in the following:

- The **X-MAC** layer [170] implemented is the first radio duty-cycled E^2-MAC protocol available for Contiki. The X-MAC protocol algorithm, as proposed by *Buettner et al.* [25], is discussed in detail in Section 2.4. In its original implementation on top of the MANTIS OS [20], X-MAC additionally applies a traffic estimation algorithm to adapt the wake-up interval to the traffic load. The current Contiki X-MAC implementation, however, does not yet support such an adaptation, the wake-up interval is statically defined at compile time and kept unchanged at run-time.
- The **ContikiMAC** [48] protocol layer is the successor of X-MAC in Contiki v.2.5 and combines key concepts used in various other E^2-MAC protocols. It is further discussed in Section 2.4. Borrowed from X-MAC, ContikiMAC relies on strobing to notify about an upcoming transmission, but directly uses data packets as strobes. With ContikiMAC, nodes learn each neighbor's wake-up schedules for saving energy in future transmissions.
- The **Low Power Probing** protocol [130] layer is a receiver-based duty-cycled E^2-MAC protocol. All nodes periodically send out probe packets indicating that they are ready to receive data. This behavior is especially beneficial for broadcasting. LPP is discussed in more detail in Section 2.4.
- The **NullMAC** protocol layer is a minimalistic protocol that just passes data packets from the network layer to the radio driver and vice versa. This implies that NullMAC does not check the radio channel for activity before sending a packet. Therefore, NullMAC should preferably be combined with Contiki's Carrier Sense Multiple Access (CSMA) layer to avoid collisions (c.f. *Boano et al.* [21]), which we also did in Chapter 9. NullMAC does not duty-cycle the radio to save energy. The resulting high energy consumption is, however, compensated with a high throughput, as clearly demonstrated in Chapter 9.

2.2 Wireless Sensor Network Testbeds

For years, simulation has been the research methodology of choice in the majority of studies on wireless ad-hoc and sensor networks. However, with discovering wide gaps between simulation results and real-world prototype results, the appropriateness of simulation tools for evaluating wireless phenomena has more and more been been questioned. Especially in the wireless sensor and ad-hoc network community, inappropriate parameter settings and unrealistic radio, traffic and/or mobility models have been identified and criticized as a general drawback of simulation studies, e.g., by *Kurkowski et al.* [105] and *Andel et al.* [11]. With the field growing more mature, researchers have generally aimed at proofing the real-world feasibility of their protocols on real-world devices. For evaluating protocol behavior in practice, experimental WSN testbeds have become indispensable today [95].

2.2.1 Existing Testbeds of Wireless Sensor Networks

In the past five years, numerous universities and research institutions have started to set up real-world sensor network testbeds. In most cases, these testbeds have been set up for research and teaching purposes, in order to enable testing and evaluation of real-world behavior of developed protocol mechanisms. An increasing number of stationary WSN testbeds have been put into operation, with different node hardware, and very heavily differing architectural testbed design. Since most testbed setups have been initiated as independent projects from single research institutions, little attention has yet been devoted to coordination and standardization of the essential testbed management functionalities. Such services comprise, but are not limited to user account management, resource reservation mechanisms, provisions for remote node reprogramming, and a consistent representation of the experiment results. The European-Union WISEBED [162] project targeted at bridging this gap by establishing an experimental federation of testbeds of WSNs with unified and standardized interfaces, all of them made accessible over the Internet to the European WSN research community. The management architecture presented in Chapter 3 forms a central part of the WISEBED software architecture.

While there are certainly many more testbeds that would deserve to be mentioned, we briefly portray and describe the most prominent wireless sensor network testbed deployments and their most important properties and characteristics.

- MoteLab [182] is a sensor network testbed on the campus of Harvard University. It currently features roughly 190 TelosB [144] sensor nodes. The nodes are wired to programming boards for reprogramming and communication.
- The TWIST testbed [75] is located in a building of TU Berlin and spans across several floors. The total number of sensor nodes belonging to the testbed is approximately 200, featuring two hardware types. The testbed is organized hierarchically in 3 tiers, consisting of servers, super nodes and sensor nodes.
- Kansei [60] is a sensor network testbed located at Ohio State University, which targets at research in large indoors sensor networks. Currently, approximately 200 sensor nodes are deployed, along with the same number of gateway stations attached to each one of the sensor nodes.
- PowerBench [76] at Technical University of Delft is a testbed infrastructure specifically designed for benchmarking power consumption. It includes hardware and software components for capturing the power traces the nodes in the testbed in parallel, which can be used for offline processing and debugging.
- TutorNet [176] at University of Southern California uses a 3-tier network topology with testbed servers, gateway stations, and sensor nodes connected via USB to the gateways. Currently, there are more than 100 nodes deployed.
- Sensei-UU [155] is a relocatable testbed designed to enable users to repeat experiments with mobile, heterogeneous nodes in diverse environments, in contrast to using a static indoor testbed with predefined hardware and sensor equipment. The testbed has been set up in different locations in and around the Uni-

versity of Uppsala campus for various research activities related to mobility.

2.2.2 Testbed Management Solutions

Most of the abovelisted testbeds are managed by simple tailormade management solutions, which allow for accessing the testbed, reprogramming the sensor nodes and receiving the experiment results, e.g., in the form of some textfiles. However, most of these management solutions have generally been tightly coupled to one particular testbed deployment and the employed hardware, and are not easily reusable for further testbed setups. We briefly portray the different management software toolkits along with their basic features.

- *MoteWeb* [128] is a simple management system including a web-interface for Harvard's Motelab. The webpage displays the list of available nodes and their current status. In a restricted area, registered users can upload executable files, assign those executables to the individual nodes to create a so-called *job*, and schedule the job to be run on MoteLab. Only one job is allowed at the same time. During the experiment run, output is streamed into the database and then made available for the user to download. However, it is not possible for the user to monitor the experiment output at run-time, neither is it possible to interact with it, e.g., by sending commands to certain nodes.
- The management software of the TWIST testbed *TWISTv1* [177] is available to the public. The software has more or less the same features as MoteWeb and is generally quite tightly coupled to the Ethernet-based wired setup of the testbed and the 3-tiered-architecture of TWIST, which use NSLU2 [117] devices with embedded Linux [134] to access and reprogram TelosB sensor nodes. The tight interaction of *TWISTv1* with specifics of these NSLU2 gateway nodes, however, puts the re-useability of the software for different testbed setups with differing architectures and node equipment into question. TWIST offers a web-based user interface to submit jobs and allows for monitoring these at run-time.
- *Kansei Software:* Besides some indications about the Kansei software in [60] stating that the Kansei software is "similar to Motelab in the underlying Open-Source Linux-Apache-MySQL-PHP/Perl implementation technology", there is little information available about the Kansei testbed management software. [60] claims that the testbed features a web-interface for submitting jobs to the testbed and retrieving results, which at the time of editing this thesis was not accessible. Real-time features such as monitoring the experiment output at run-time and real-time user interaction are not known to be supported.
- The software of the TutorNet [176] testbed consists of a web-interface that interacts with their 3-tier topology of TelosB motes and intermediate gateway nodes. Authenticated users can connect to the testbed servers and then use command-line tools to control the testbed nodes. Real-time features such as monitoring the experiment output at run-time and real-time user interaction are, however, not known to be supported as well.

Thoroughly studying the related work on today's existing wireless sensor network testbeds, we come to the conclusion that without any doubt, a lot of work has been dedicated to the setup of large and complex WSN testbeds, as well as for the design and implementation of tailormade software for network administration tasks. However, although the software solutions for managing and administering experimentation on these testbeds all fulfill similar or equal functionalities, none of the encountered solutions offers is *independent* from the sensor node type or operating system or certain architectural assumptions (e.g., a 3-tiered-topology with Ethernet backbone). A consistent separation of the management functionality into components with clearly defined interfaces, such as user administration, resource reservation, experiment execution does not exist. Hence, all of the surveyed solutions exhibit a lack of *portability* and *reusability*, since no standardized off-the-shelf components have evolved. Therefore, it is hard for other research groups intending to set up an own WSN testbed with a different kind of sensor nodes, since often, basic testbed functionalities have to be (re)implemented from scratch.

We bridge this missing gap with the TARWIS testbed management architecture presented in Chapter 3, which relies on cleanly separated modular components. TARWIS has to date been successfully integrated with nine different WSN testbeds throughout Europe, all of them with heavily differing architectural design and equipped with heterogeneous sensor nodes and operating systems.

2.3 Energy Measurement and Estimation Techniques

This section discusses the most frequently applied hardware-based physical measurement methodologies in the context of wireless sensor network studies, namely the application of Digital Storage Oscilloscopes (DSOs) in Section 2.3.1 and the usage of dedicated measurement devices in Section 2.3.2. Throughout the course of this thesis, we have made use of the DSO-based methodology mainly for the purpose of debugging and inspecting protocol behavior, since the current trace displayed in DSO's is a convenient form of verifying that the current code implementation behaves as intended, and allows for tracking down bugs.

Thanks to our collaboration with the Karlsruhe Institute of Technology (KIT), which developed an own dedicated measurement hardware (cf. Section 2.3.2), we could conduct the majority of our measurements with the latter platform, which turned out to be much more comfortable to operate, and which permitted us to make continuous measurements over longer timespans, e.g., 10-15 minutes or even more, as opposed to maximum 60 seconds with our HAMEG HM1508-2 DSO [73].

2.3.1 Digital Storage Oscilloscopes

An oscilloscope is an electronic test instrument that permits the observation of varying signal voltages. Oscilloscopes usually plot a two-dimensional graph of electrical potential differences as a function of time. Digital Storage Oscilloscopes

Figure 2.5: HAMEG Oscilloscope

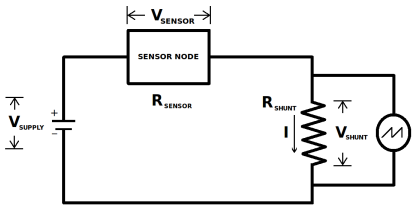


Figure 2.6: Measurement Circuit

(DSO) further supply the obtained measurements as digital data streams. This methodology has been applied to quantify the energy consumption of sensor nodes, cellphones or mesh nodes in numerous studies, e.g., *Feeney et al.* [63] or *Polastre et al.* [143].

We made use of the HAMEG HM1508-2 [73] digital oscilloscope (cf. Figure 2.5) mainly for preliminary evaluations and for debugging MAC protocols on sensor nodes. The software HMLab 1.06 permits to digitally read out the measurements from the DSO with a sampling rate of up to 1 GS/s (= 10^9 samples per second). Figure 2.6 depicts the circuit used for measuring the small current of a sensor node. A sensor node is sourced by a mains adapter and connected in series with a 1Ω shunt resistor. The current flowing through the resistor is proportional to the resistive voltage drop across the shunt, which can be measured using the DSO. With knowing the resistance of the shunt in advance, the timely varying current flowing through the measurement circuit can hence be derived. High-precision shunt resistors with a resistance of 1Ω usually exhibit a tolerance (=maximum deviation from the indicated value) in the range of $\pm 1\%$ to $\pm 1‰$.

By measuring the voltage drop across the shunt resistor, we obtain $V_{shunt}(t)$. The measured voltage then varies with the different operation modes of the components on the board of the sensor node (radio on/off, CPU frequency, LED state, etc.). Applying Ohm's law (2.1) leads to the current draw over time $I(t)$ that flows through the measurement circuit (2.2).

$$V = R \cdot I \tag{2.1}$$

$$I(t) = \frac{V_{shunt}(t)}{R_{shunt}} \tag{2.2}$$

To calculate the instantaneous power consumption of sensor node node $P_{sensor}(t)$, we apply the power relationship (2.3)

$$P = V \cdot I \tag{2.3}$$

as well as Kirchhoff's mesh rule, according which the sum of all electrical potential differences around any closed circuit must be zero (2.4).

$$\sum_{k=1}^{n} V_k = 0 \tag{2.4}$$

Applying equation (2.4) to the measurement circuit of Figure 2.6 yields

$$V_{supply} = V_{sensor} + V_{shunt}$$

The instantaneous power consumption of the sensor node hence calculates as

$$P_{sensor}(t) = (V_{supply} - V_{shunt}(t)) \cdot I(t)$$

With V_{shunt} being very small compared to V_{supply} ($< 1‰$), we can simplify the above relationship to

$$P_{sensor}(t) = V_{supply} \cdot I(t) = V_{supply} \cdot \frac{V_{shunt}(t)}{R_{shunt}} \tag{2.5}$$

The energy consumed by the sensor node over a time interval $[t_{start}, t_{end}]$ is the integral of power consumption over time.

$$E_{sensor_{[t_{start},t_{end}]}} = \int_{t_{start}}^{t_{end}} P_{sensor}(t)\, dt$$

Nesting the expression $P_S(t)$ of equation (2.5) into the above formula yields

$$E_{sensor_{[t_{start},t_{end}]}} = \frac{V_{supply}}{R_{shunt}} \int_{t_{start}}^{t_{end}} V_{shunt}(t)\, dt$$

Using the numerical values of $V_{shunt}(t)$, the energy consumed by the node over in a time interval $[t_{start}, t_{end}]$ can thus be numerically approximated as

$$E_{sensor_{[t_{start},t_{end}]}} = \frac{V_{supply}}{R_{shunt}} \sum_{t_{start}}^{t_{end}} V_{shunt}(t)\, \Delta t$$

where Δt equals the inverse of the sampling frequency (e.g., Δt = 1 ms for a sampling frequency of 1000 Hz).

In the current trace measured with a DSO depicted in Figure 2.7, the energy consumed over 1 s hence corresponds to the area below the curve, times the voltage.

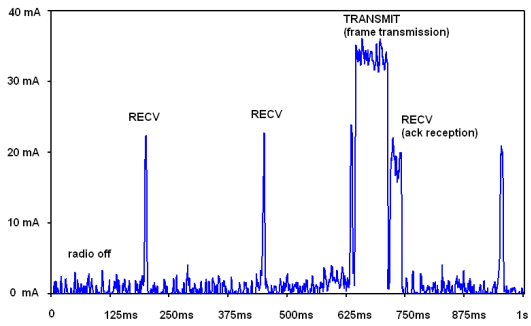


Figure 2.7: DSO-generated Current Trace of the WiseMAC Prototype

2.3.2 Sensor Network Management Devices

The Sensor Node Management Device (SNMD) [77] depicted in Figures 2.8 and Figure 2.9 has been developed by the Karlsruhe Institute of Technology (KIT). The KIT has equipped its WSN testbed with these precise but cost-efficient hardware-based energy measurement components. SNMDs have been specifically designed to accurately measure currents and voltages of sensor nodes with a sampling resolutions of up to 20 kHz, or even up to 500 kHz in the so-called buffered mode.

In analogy to the methodology described in Section 2.3.1, SNMDs measure the resistive voltage drop across a 1 Ω shunt resistor. This voltage is then converted to a raw value using an A/D converter, which is fed to the onboard Atmel microcontroller. The SNMD firmware allows for reading out this value with different sampling rates over an USB interface. Each SNMD has been calibrated using high-precision laboratory equipment for different current ranges. The SNMD firmware corrects each sampled measurement by an error term, which was obtained during evaluative testing in advance. This has been shown by *Hergenroeder et al.* [78] to reduce the measurement error introduced by the measurement circuit below $\pm$ 0.5% for any current in the range of 0-100 mA, an accuracy range which is definitely sufficient to rely any experimental and comparative analysis of sensor network mechanisms upon.

The energy consumed by the sensor node within any time interval $[t_{start}, t_{end}]$ is the integral of its power consumption $P_{sensor}(t)$ over time.

$$E_{sensor_{[t_{start},t_{end}]}} = \int_{t_{start}}^{t_{end}} P_{sensor}(t)\, dt$$

Using the power relationship $P_{sensor}(t) = V(t) \cdot I(t)$ and the numerical values of $I(t)$ and $V(t)$ obtained by the SNMD, the energy consumed by the node over a time interval $[t_{start}, t_{end}]$ can be numerically calculated as

$$E_{sensor_{[t_{start},t_{end}]}} = \sum_{t_{start}}^{t_{end}} V(t) \cdot I(t)\, \Delta t$$

where Δt equals the inverse of the sampling frequency (e.g., Δt = 1ms for a sampling frequency of 1000 Hz).

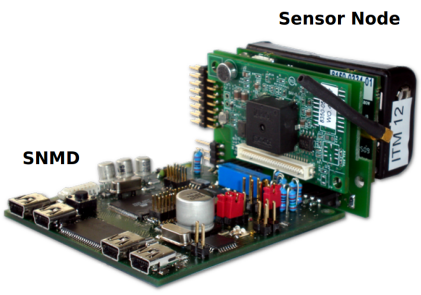


Figure 2.8: SNMD with Node attached

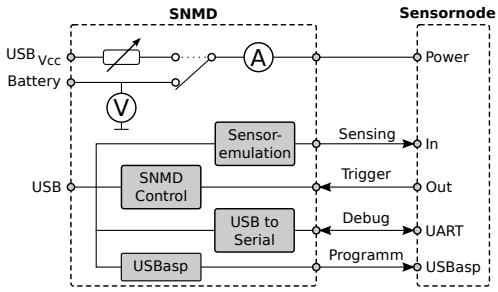


Figure 2.9: SNMD Block Schema [78]

2.3.3 Software-based Energy-Estimation

Software-based mechanisms to estimate the energy consumption of sensor nodes have yet often been applied in experimental WSN studies. In order to quantify the energy consumption of a WSN testbed, it is often impractical, time-consuming and probably too costly to hook every single node to a measurement device. Simple state-based energy-estimation models have already been used in the prominent studies on S-MAC [190] and B-MAC [143]. However, as in most related studies, the authors neglected to evaluate the *accuracy* of their energy estimation model.

Dunkels et al. [50] motivate the need for software-based *on-line* energy estimation, because only *on-line* estimation mechanisms running on the node itself enable the node to take energy-aware decisions about routing, clustering or transmission power scheduling. The authors derive the state-based model (2.6) and experimentally correlate the estimated energy with the sensor nodes lifetime.

$$E = (I_m t_m + I_l t_l + I_r t_r + \sum_i I_{c_i} t_{c_i}) \cdot V \tag{2.6}$$

In this energy model, V is the supply voltage, and I_m, t_m are the current draw of the node's microchip and the time it has been fully active. The variables I_l and t_l correspond to the current draw and time of the microchip in the low power mode. Variables I_t and t_t correspond to the current draw and time of the radio transceiver in the transmit mode, and I_r and t_r in receive mode. Furthermore, I_{c_i} and t_{c_i} denote current and time of operation of further onboard components. Having evaluated their software-based model (2.6) in a range of experiments, the authors of [50] note that their estimations are "sound" and that they correlate with the measured node's lifetimes, however underline that "further study is needed to accurately quantify the error rate of the mechanism".

In their PowerBench study, *Haratcherev et al.* [76] elaborate on the difference between their software-based energy estimations and the physically measured energy consumption of sensor nodes running different energy-efficient MAC protocols. The model (2.7) applied is the same as that of the studies on S-MAC [190] and B-MAC [143]. The consumed energy E is calculated as the sum of the total time spent in the receive state multiplied by the respective power level $T_{rcv}P_{rcv}$, and the respective terms for the transmit and sleep states ($T_{slp}P_{slp}$ and $T_{tx}P_{tx}$).

$$E = P_{rcv}T_{rcv} + P_{tx}T_{tx} + P_{slp}T_{slp} = (I_{rcv}T_{rcv} + I_{tx}T_{tx} + I_{slp}T_{slp}) \cdot V \tag{2.7}$$

When running B-MAC [143] and Crankshaft [71], this difference reaches up to 21% of the measurement values. *Per-node calibration* is shown to vastly reduce this estimation error. With the deviations between software-based estimation and physical measurements still ranging from 2% to almost 14% for some of the examined E^2-MAC protocols, the software-based estimation approach still leaves room for further improvements. The authors further note that frequency of state transitions have a significant impact on the estimation accuracy.

2.4 Medium Access Control

In order to make sensor networks a viable instrument for a broad range of real-world (including outdoor) applications, the employed nodes need to satisfy various constraints. Besides the limitations with respect to size, costs and physical robustness, one of the major constraints is that of remaining operable without a wireline infrastructure for a reasonable amount of time, preferably weeks or months. Since most sensor nodes are initially equipped with some few AA batteries, reaching such a lifetime is only possible if the node keeps its energy consumption limited to a few milliamperes. Keeping all onboard sensors and radio interfaces constantly turned on, most platforms drain out of power within not much more than a few days, as recently pointed out by *Nguyen et al.* [132]. Intelligent mechanisms to make use of the resources in an energy-efficient manner are hence required to reduce the overall power consumption and hence prolong the sensor nodes' lifetime.

2.4.1 Challenges in Medium Access Control Design

The onboard sensors for measuring physical values (e.g. temperature, humidity, luminosity) can easily be operated in an energy-efficient manner. By turning them on just before every sensing operation, and likewise turning them off afterwards, their energy consumption can be minimized and no energy is wasted without use. However, the radio transceiver can not be handled in the same manner, since communication of data between two radio transceivers requires two of them being active at the same time - one radio transceiver *transmitting* and the other radio transceiver *receiving*. Naturally, a transmission attempt to a targeted receiver will remain unnoticed if its radio transceiver is turned off at that time. Since nodes should permanently uphold a connection to their neighboring nodes in order to handle incoming traffic or be able to forward own sensed data, mechanisms had to be designed to achieve the two goals at the same time: keeping the network connected *but* having the radio transceiver turned off for most of the time.

This problem has generally been addressed in the last decade with the development of various *Energy-Efficient MAC (E^2-MAC)* protocols. Generally speaking, E^2-MAC protocols are a class of distributed algorithms running on the individual sensor nodes, which solve the communication problem *and* at the same time satisfy the sensor nodes' inherent constraints with respect to energy-efficiency by *duty-cycling* the radio transceiver. An E^2-MAC protocol typically addresses the following problems, which are inherently raised by the communication problem itself, the duty-cycling of the radio transceiver or the nature of the wireless channel:

- **Synchronicity**: With nodes turning off their radio for most of the time, an E^2-MAC protocol must ensure that, when communication takes place, both the transmitting node and the receiving node have their radio transceivers *synchronously* turned on and in the correct mode. Various approaches have been proposed throughout the last decade to reach this behavior: the proposed techniques range from maintaining a synchronous sleep-wake schedule, where nodes

exchange traffic or traffic-announcements, over busy-tone schemes to *catch* the targeted receiver in a wake-up, to using external low-power *wake-up radios* operating on a different channel, signaling the receiver to turn on its main radio before transmitting the data over the main channel.

- **Collision Avoidance**: Most sensor node platforms only use one low-bandwidth channel, in which they have to communicate in half-duplex mode. Their radio transceiver permits them to either *transmit* or *receive*, but not both at the same time. If two nodes transmit at the same time, receivers in their transmission range will only receive noise, which renders both transmissions useless and makes retransmissions necessary. E^2-MAC protocols usually integrate mechanisms ascertaining that this situation does not or only rarely occur. Some even attempt to mitigate collisions that can not be detected at the transmitter alone by checking the channel before transmission (i.e., the *hidden node* problem).
- **Fairness**: E^2-MAC protocols have to make sure that the wireless channel is allocated fairly among the participating nodes in the network, such that all nodes can access the channel within reasonable access time. It is generally desirable that, given that the channel is a limited resource, all nodes in the network can deliver a similar share of their packets to the sink.
- **Latency**: With the radio transceiver alternating between wake and sleep periods, nodes often have to buffer pending packets and *wait* for the next transmission opportunity. This increases the latency between packet generation at a leaf node, and packet arrival at the sink, especially if the packet has to travel across several hops. Therefore, E^2-MAC protocols generally apply short intervals with which the radio transceiver is duty-cycled, usually in the magnitude of a few 100 milliseconds. Nevertheless, duty-cycling the radio inherently increases the latency, compared to energy-unconstrained MAC protocols.
- **Throughput**: The situation that more than one packet is buffered and has to be transmitted to the same receiver is rather frequent in WSNs, e.g., because the data to be transmitted is too large to fit into one frame. E^2-MAC protocols therefore usually integrate solutions for letting senders transmit several packets in a row to targeted receivers in so-called *burst transfers*. Although this is an effective means to transmit a rather large portion of data across one link, the maximum achievable throughput compared to energy-unconstrained wireless channel MAC protocols is still much degraded, especially over multiple hops.

The radio transceiver unfortunately has turned out to be the most power-hungry component on most of today's node platforms, which again highlights the importance of efficient and robust E^2-MAC protocols. With the radio constantly turned on, a customary TelosB node powered with two AA batteries depletes within a few days, which is insufficient for many WSN applications. Applying state of the art E^2-MAC protocols with duty-cycles in the range of a few percent, a node's lifetime can be pushed to the order of a few weeks or months. This value, however, heavily depends on the volume of the received and transmitted traffic and the computational load of the CPU, as demonstrated in *Nguyen et al.* [132].

2.4.2 Sources of Energy Waste

The E^2-MAC protocols designed throughout the last decade generally attempt to reduce the major sources of *energy waste* of wireless channel MAC protocols. There's a widespread agreement on what the major sources of *energy waste* on the MAC layer are. Many studies have pointed out the potential for energy savings in wireless channel protocols, e.g., in the case of IEEE 802.11-based networks with access-point-based scenarios (cf. *Anastasi et al.* [10]) or in (mobile) ad-hoc network scenarios (cf. *Zheng et al.* [192]). In the context of WSNs, *Ye et al.* [189] identified the main sources of energy waste to be *Idle Listening*, *Overhearing*, *Collisions* and *Protocol Overhead*.

- **Idle Listening** refers to keeping the radio in the receive state in order to wait for incoming traffic. The communication problem between any sender and receiver generally consists in the information asymmetry concerning the message to be exchanged *and* the time of transmission. Since nodes need to have their radio turned on when the sender transmits the message, but can inherently not predict the time of transmission, waiting periods where nodes have the radio turned on without actually receiving data are unavoidable. As receiving data and listening to an idle channel are almost equally expensive on most radio transceivers, idle listening often constitutes a major source of energy waste.
- **Overhearing** refers to the reception of data frames that are not destined to the receiving node. Since the wireless communication in wireless sensor networks is of broadcast nature on the physical layer, frames will be received by all nodes in the transmitter's transmission range. [189] identifies *idle listening* and *overhearing* to be the two topmost reasons for energy waste.
- **Collisions** of transmissions occur when two nodes concurrently use the same channel by transmitting frames to the same or other nodes in the vicinity. By introducing fine-grained rules for the medium access, e.g., the coordination of a collision-free transmission schedule or contention mechanisms, collisions can generally be reduced, but rarely fully prevented. In case of a collision, nodes spend a significant amount of energy for the frame transmission itself and for keeping the transceiver ready to receive an acknowledgement. Usually, MAC protocols employ costly retransmission schemes in order to recover collisions.
- **Protocol Overhead** is a frequent source of energy waste in wireless channel MAC protocols. Some protocols require continuous and excessive signalization, e.g., to maintain a collision-free schedule and a network-wide time-synchronization. Since these message exchanges do not contain any application payload data, they must be considered overhead.

2.4.3 A Taxonomy of Wireless Sensor MAC Protocols

In the past decade, a large number of energy-efficient MAC protocols for wireless sensor networks have been proposed. Almost every wireless channel multiplexing technique (i.e. SDMA, TDMA, CDMA, FDMA) and every medium access

scheme has been evaluated. Since most of the sensor node platforms today are equipped with one single low-power single channel radio transceiver, we focus in this section primarily on related work in the area of single-channel MAC protocols. The protocols proposed and used in single-channel wireless sensor networks differ in the manner how the nodes organize the access to the shared radio channel. *Langendoen* [109] distinguishes three classes of organization in today's wireless channel MAC protocols: *frame-based access*, *slotted access*, and *random access* protocols. We briefly discuss the main characteristics of these protocol classes with a few examples, starting from the most complex and most intensively provisioned frame-based protocols moving towards the least complex class of the random access protocols. Each protocol that was implemented and/or evaluated in one of the subsequent chapters (in simulation and/or on a real-world testbed), is schematically illustrated, cf. Figures 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, and 2.16.

Finally, we briefly discuss three protocols of the class of *wake-up radio-based medium access* protocols, which follow a totally different concept of energy-aware medium access control than the three classes discussed on the following.

Frame-based MAC Protocols

The class of *frame-based* MAC protocols pursues a TDMA-approach and generally requires a network- or at least clusterwise synchronization scheme. Transmissions take place during time slots that are allocated to single nodes or links. Slots for single nodes can either be defined for transmission or reception, be segmented to allow both directions (using disjoint windows), or be random-access/contention-based inside the slots. Frame-based protocols have several inherent advantages compared to the other classes of protocols. First, by partitioning the time into slices and allocating them to senders and receivers, nodes can turn off the radio when their slot is over. Second, collisions due to concurrent channel access are prevented to a large extent, since the fine-grained scheduling of the channel prevents nodes from attempting to access the channel at the same time. Third, minimal Quality of Service and fairness can be *guaranteed*, which is a major advantage over contention-based protocols.

Problems of frame-based MAC protocols generally arise when determining how the slots are assigned to the nodes, since in sensor networks, there is usually no central node administering the slot allocation. Since the MAC protocol must prevent that overlapping time slots lead to concurrent medium access and collisions, frame-based MAC protocols have a continuous need for rigid time synchronization. It has been observed many times that the cost for maintaining a collision-free schedule over a long period of time can easily outweigh the energy spent for the actual data traffic. Another problem is that the resulting medium access scheme becomes rather inflexible and may not correspond to the shape and the volume of traffic generated by the application. While some nodes may need more slots for transmission, many slots may be wasted for idle senders. This resulting lack of flex-

ibility is often referred-to as *overprovisioning*. TRAMA [151] and LMAC [151] are two of the most well-known protocols of this class.

In the **Traffic-Adaptive Medium Access Control Protocol (TRAMA)** [151], the nodes regularly broadcast information about traffic flows as well as their local neighborhood. By observing these reports, a node learns the identities of all its 2-hop neighbors. These identities are used to determine a collision-free schedule by using a hash function that takes the node identifier and the number of the current slot as arguments. By taking into account the 2-hop neighborhood, transmissions are protected against hidden node interference.

The **Lightweight MAC (LMAC)** [33] protocol is similarly based on distributing slots and taking into account the 2-hop neighborhood of the transmitting node. The reduction of the costly radio state transitions, during which the radio can neither receive nor transmit, is a major design goal of LMAC. The protocol hence renounces on intermediate frame acknowledgements after each frame-reception to avoid more radio switches, and thereby shifts issues related to reliability to the upper layers.

Slotted Access MAC Protocols

In *slotted access* protocols, nodes are synchronized to a common sleep-wake pattern. Nodes wake up at the beginning of each *slot* to exchange pending traffic or traffic announcements. In contrast to frame-based access MAC protocols, the time inside one slot is not uniquely assigned to nodes or links. No collision-free schedule is calculated and assigned by central entities. The common sleep/wake pattern is only utilized as *rendez-vous* scheme, which makes this class of protocols much less complex than frame-based protocols. Collision avoidance is usually implemented by contention inside each slot. The probability of collisions is much higher than, e.g., with random access protocols, since all nodes attempt to transmit during the brief designated common wake-up times at the beginning of each slot. A further drawback of this class of protocols is the high overhead of the common sleep-wake pattern: nodes constantly need to maintain a network-wide synchronized wake-up schedule, which makes continuous SYNC message exchanges necessary. Given that many monitoring sensor network applications generate only a few packets per hour or even per day, this cost can not be considered negligible.

The **Sensor-MAC (S-MAC)** [190] protocol is the most prominent protocol of this kind. S-MAC synchronizes the wake-ups of the nodes in so-called synchronization *clusters*. In each wake-up slot, nodes stay awake for an active window of fixed duration, as displayed in Figure 2.10. Nodes regularly broadcast SYNC packets at the beginning of a slot, such that neighboring nodes receiving them can adjust their clocks to the latest SYNC. Nodes joining the network first listen for SYNC packets in order to follow to the propagated wake-up pattern of a synchronization cluster. When not hearing any SYNC messages, they start alternating in their own wake-up pattern and propagate it with own SYNC messages, hoping that other nodes will join their cluster. When multiple clusters evolve in a multi-hop network topology,

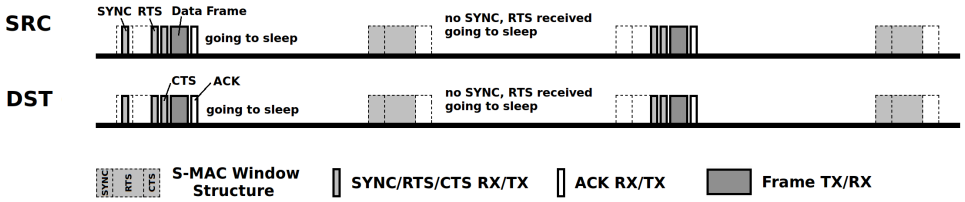


Figure 2.10: Sensor MAC Protocol (S-MAC)

bordering nodes between two clusters adopt the wake-patterns of both clusters, in order to provide interconnecting links. Before every data frame transmission, S-MAC applies an RTS/CTS handshake for collision avoidance. Due to the fixed duration of the *active window*, S-MAC has been shown to be rather inflexible under variable load in several studies, e.g., *van Dam et al.* [38] or our recent study [90].

The **Timeout-MAC (T-MAC)** [38] protocol has been proposed to enhance S-MAC under variable load. The listening interval in T-MAC is is extended if nodes perceive signs of pending transmissions. It ends when no so-called *activation event* has occurred for a given time threshold, which is much shorter than the fixed listen interval in S-MAC. Collision avoidance is realized using the same RTS/CTS handshake. As depicted in Figure 2.11, a packet can travel at least two hops per wake-up interval. When node SRC transmits a packet to DST, the next node in line $DSTII$ overhears the CTS and is able to respond to the subsequent RTS. An optional scheme is proposed to solve the so-called *early sleeping problem*. This problem arises when a node intends to transmit a message across three links $A \rightarrow B \rightarrow C \rightarrow D$, with node D going to sleep *too early* because it has not been notified of an upcoming transmission by an RTS. The solution consists in node C sending a *Future Request to Send* packet to node D to keep it awake.

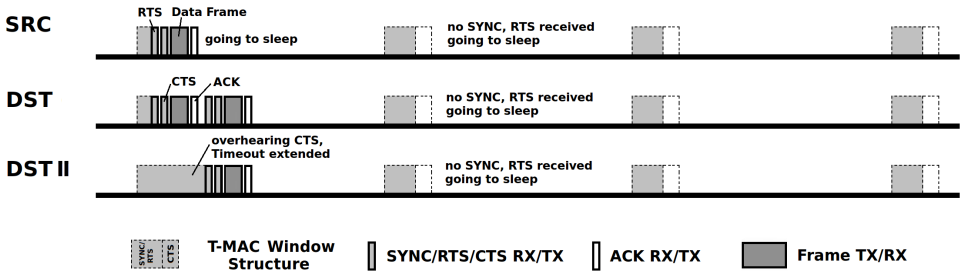


Figure 2.11: Timeout MAC Protocol (T-MAC)

Random Access MAC Protocols

Random access protocols are the most simple class of MAC protocols. They are inspired by classical *best-effort* CSMA/CA protocols and neglect the need for costly (over-) provisioning of the channel through complex distributed algorithms. Instead, nodes access the channel whenever there are packets to be sent, sometimes taking into account the wake schedules of the targeted receivers. *Random backoff*

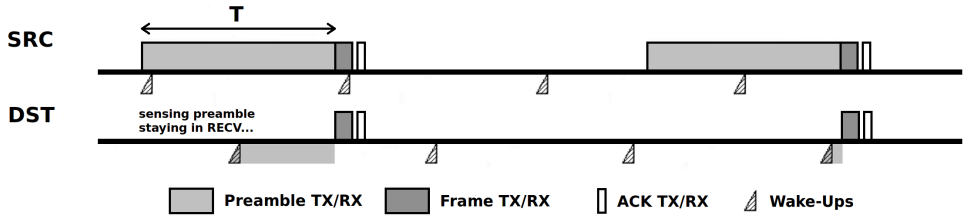


Figure 2.12: Berkeley MAC Protocol (B-MAC)

contention mechanisms are usually employed to avoid collisions with other nodes intending to transmit at the same time, which does reduce but not eliminate the collision probability. Random access protocols do not impose a common wake-up schedule for the entire network or node clusters, hence there is no need for synchronizing the nodes' clocks. The low complexity makes this class of protocols rather inexpensive with respect to the energy spent for the control message overhead. In case of no data traffic, random access protocols usually exhibit the lowest energy footprint. Another advantage of this class the high flexibility: since the medium is not pre-allocated to certain nodes or links, random access protocols generally support variable load patterns better than slot-based or frame-based protocols. However, the challenge with random access contention-based protocols often consists in finding ways to reduce the energy waste caused by collisions and overhearing, and to some degree also idle-listening. Furthermore, random access protocols can suffer heavily from congestion. Under elevated load conditions, Quality of Service and fairness can usually not be guaranteed. The most prominent protocols of this class are the Berkeley-MAC (B-MAC) [143] protocol, the Wireless Sensor MAC protocol (WiseMAC) [57] and the X-MAC [25] protocol, which we briefly discuss in the following, among a selection of other random-access MAC protocols that influenced our contributions and/or were used as reference protocols in this thesis.

The **Berkeley-MAC (B-MAC)** [143] protocol depicted in Figure 2.12 employs a constant static wake-up pattern, where the node briefly turns on its radio in each interval to poll the channel, keeping it turned on if the busy tone is heard, and turning it off if nothing is received. Since checking the channel can be done very quickly with most radio transceivers, the time the radio is turned on can be minimized to a few milliseconds. This allows for very low idle duty-cycles, i.e., the ratio between the radio receive time and the radio sleep time. Often, duty-cycles are only at 1% or even below. When attempting to transmit a frame, a wake-up tone is sent during the entire wake-up interval to alert receivers for the upcoming frame transmission. This techniques is often referred-to as *Low-Power-Listening (LPL)* or *Preamble Sampling*. B-MAC is one of the most widely used protocols, especially in deployment studies and real-world applications. To date, it is the default radio duty-cycling MAC protocol in the TinyOS operating system [112]. Unlike our contribution in Chapter 7, the B-MAC protocol neglects to take into account knowledge of the wake-up schedules of its neighbors nor the currently encountered traffic rate to adapt any of its parameters at run-time.

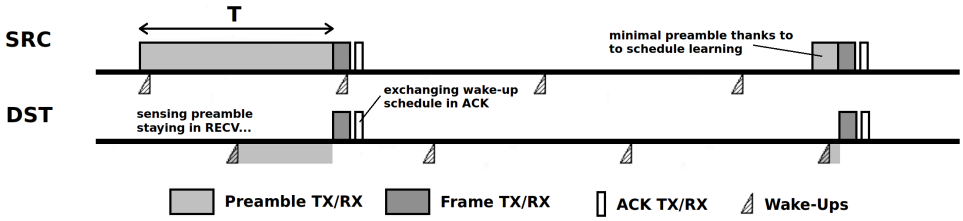


Figure 2.13: Wireless Sensor MAC Protocol (WiseMAC)

The **Wireless Sensor MAC (WiseMAC)** [57] protocol shares many similarities to B-MAC. It also consists of short periodic duty-cycles with a statically configured interval to sense the channel for the presence of a preamble signal, a busy tone that alerts nodes to stay awake for the upcoming frame transmission. As in B-MAC, nodes sample the medium with the same interval, with sampling schedules left independent and unsynchronized. A preamble is sent before each frame transmission to alert the receiving node in its wake-up. In contrast to B-MAC, WiseMAC *learns* the wake-up schedules of neighboring nodes by exchanging the schedule offset in the frame header and the acknowledgement. WiseMAC saves these schedule offsets in its schedule offset table, and henceforth minimizes the length of the preambles in future transmissions, which notably pays off with respect to the energy consumption. Figure 2.13 depicts this mechanism with the second transmission, which only compensates for the clock drift, and adds some small random duration for the medium reservation mechanism.

The **X-MAC** [25] protocol introduces a series of short preambles before every packet transmission, allowing the receiver to reply in between with a so-called *Early-ACK*. On average, it thereby halves the preamble transmission costs compared to B-MAC, assuming uniform distribution of the sampling schedules over time. Each preamble strobe contains target address information, which limits the overhearing problem: non-targeted receivers simply turn off the radio and wait for the next wake-up whenever they receive a preamble strobe containing a different identifier. Eversince its implementation [170] in the Contiki operating system [47], X-MAC is one of the most frequently used protocols in the WSN community. Many deployment studies have since then used X-MAC as the default MAC protocol for Contiki on the link layer. Figure 2.14 depicts the strobing mechanism and the *Early-ACK* indicating reception readiness for the frame transmission.

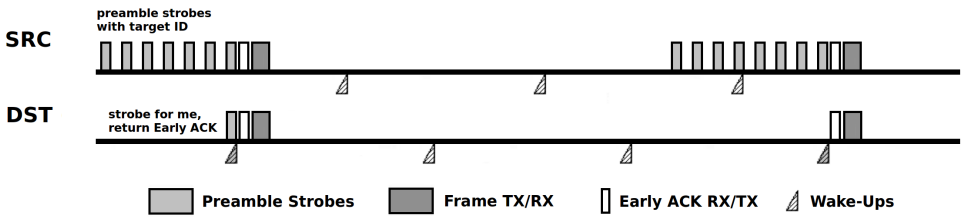


Figure 2.14: X-MAC Protocol (X-MAC)

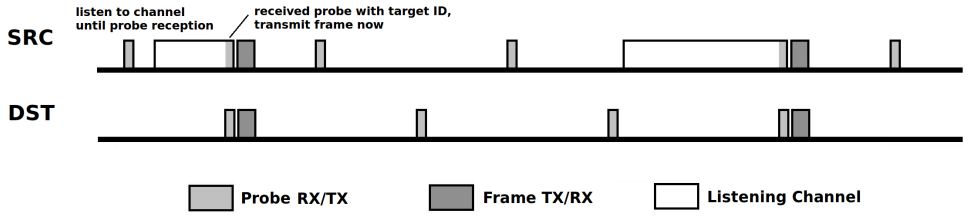


Figure 2.15: Low-Power-Probing MAC Protocol (LPP)

The **Low-Power-Probing (LPP)** [130] protocol introduces a receiver-initiated form of duty-cycling. In contrast to WiseMAC, B-MAC, and X-MAC, where the sender nodes announce frame transmissions to the targeted receivers using preambles and busy-tones, the receiving nodes announce reception-readiness by transmitting a small beacon probe frame in each wake period. If no subsequent frame is received within a short timeout after the beacon, nodes turn off their radio until their next wake period. Figure 2.15 depicts both nodes sending out beacon probes. When attempting to transmit a frame to DST, the node SRC listens to the channel to receive DST's probe, and then immediately starts the frame transmission. A similar approach is followed in the *Receiver-Initiated MAC (RI-MAC)* [187] protocol.

The **Burst-Aware Energy-Efficient Adaptive MAC (BEAM)** [12] protocol is an improvement of the Contiki X-MAC protocol and was designed to operate under variable load conditions. The integration of explicit acknowledgments allows for supporting hop-to-hop reliability, lowering the overall energy expenditures of the network for end-to-end transmissions. This makes the protocol particularly viable for TCP/IP-based communication scenarios.

ContikiMAC [48] depicted in Figure 2.16 is the successor of X-MAC as the default MAC layer in Contiki v.2.5 and combines key concepts developed in various preceding MAC protocols. Nodes wake up within static intervals to check the channel for activity. Borrowing from X-MAC, ContikiMAC uses strobed preambles to notify the receiver about an upcoming transmission. However, instead of a simple bit sequence containing only the target address, ContikiMAC sends entire data packets as strobes. Nodes turning on their radio and detecting an ongoing transmission stay awake to receive one complete packet and return an acknowledgment. Building upon the schedule exchange mechanism of WiseMAC, ContikiMAC learns the neighboring node's wake-up patterns and saves them in a schedule offset table in order to minimize the strobe transmission time in the future.

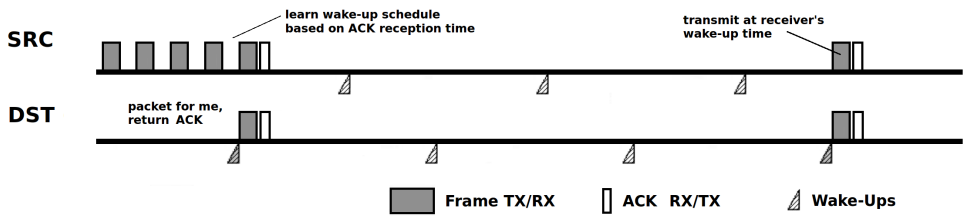


Figure 2.16: The ContikiMAC Protocol (ContikiMAC)

The above list of examples of approaches from the classes *frame-based*, *slotted access* and *random access* MAC protocols could be continued for almost arbitrarily long. *Langendoen*'s *MAC Alphabet Soup* [108] is a web-based compendium that lists and briefly characterizes the most widely cited protocols. Since protocol schemes from each individual protocol class have been thoroughly evaluated and improved in the last decade, *Langendoen* [109] conjectures that the future trend in MAC protocol design will merely consist in combining different medium access control schemes in order to create *hybrid* protocol approaches. Such hybrid approaches shall combine the advantages of two classes, e.g., to alleviate the over-provisioning problem of frame-based protocols by combining them with random access patterns. Another hybrid approach could, e.g., improve the collision susceptibility of random access protocols by integrating guaranteed slots.

With our contribution in Chapter 7, we have not followed the strategy of mixing two protocol classes, but have combined two widely used protocols with opposite goals of the same *random access* class, in order to master variable traffic load with a minimum of Quality of Service degradation. The basic operation principles such as periodic wake-up schemes based on preamble sampling offer little room for improvement, especially with respect to further increases in the (idle) energy-efficiency. Room for improvement, however, still lies in alleviating the artificial restrictions imposed by most of the classical E^2-MAC protocols, namely the heavily restrained maximum throughput, packet delivery rate and reliability, as well as the significant increase in packet latency. E^2-MAC protocols should *ideally* avoid the well-known sources of energy-waste (e.g., idle-listening, overhearing or collisions), but only to the point where the throughput and latency requirements of the upper layers can still be met.

2.4.4 Traffic-Adaptive Medium Access Control

Energy-efficient MAC protocols duty-cycling the radio transceiver generally trade off the targeted increase in *energy-efficiency* versus classical Quality of Service parameters, in particular throughput, latency and reliability. With the radio transceiver being the most power-hungry component on most WSN platforms, researchers have focused almost uniquely on minimizing this particular source of power consumption, leaving other design goals aside. As a consequence, many of today's state of the art E^2-MAC protocols discussed beforehand are able to deliver little amounts of data at a much reduced energy cost, compared to energy-unconstrained wireless channel MAC protocols. However, the price to be paid is that of severe restrictions with respect to the maximum achievable throughput and the significant increase in packet latency. Regrettably, most E^2-MAC protocols fail to *adapt* to varying traffic load, since their most crucial parameters (e.g., wake-up interval, duty-cycle) are statically set at compile-time and remain independent of external parameters encountered at run-time, e.g., the traffic volume.

With applications of sensor networks growing more diverse, the issue of E^2-MAC protocol adaptivity with respect to variable traffic load has become an increas-

ingly important issue. Many applications have further been observed to exhibit a heavily variable load over time, which is yet difficult to master for many MAC protocols. Researchers have hence started to combine different MAC approaches, in order to combine advantages of different protocols or to alleviate the performance-degrading impact of some E^2-MAC protocols. A couple of studies therefore has studied run-time adaptive MAC protocol behavior in the past few years. The term *adaptivity* has been used eversince as an ambiguous, but popular buzzword in many WSN studies, without a clear notion of how to assess or measure *traffic adaptivity*. In this section, we understand *traffic adaptivity* in the context of E^2-MAC protocols as *the ability of the MAC protocol to dynamically and autonomously react to changing traffic requirements with (de)allocation of the respective resources needed to handle the imposed traffic with equally well Quality of Service at run-time*. This attempt to define the property still leaves room for improvement, which we specifically target with our definition of a formal notion of a metric in Chapter 6.

In the following, we discuss several concepts in today's literature on E^2-MAC protocols that tackle the design goal of *traffic adaptivity*. The protocols described in the following all relate to our contributions on traffic-adaptive energy-efficient MAC protocol design by sharing the goal of reaching a higher adaptability with respect to variations in the traffic volume at run-time, and to achieve a sound maximum throughput and acceptable latency. Half of these contributions appeared simultaneously or even later than MaxMAC in its initial appearance in [85] (i.e., the LWT-MAC [28], BEAM [12] or ZeroCal [124]).

In the **Timeout-MAC (T-MAC)** [38] protocol, an increased traffic adaptivity of the S-MAC [190] protocol is achieved by prolonging the duty-cycles of the nodes when so-called *activation events* occur. An activation event may be the sensing of any communication in the neighborhood, the end of the own data transmission or acknowledgement, the overhearing of RTS or CTS messages, which may announce further packet exchanges. However, nodes do not communicate extended wake periods: if the sender receives a CTS in the transmission from DST to $DSTII$ in Figure 2.11, it knows that DST has remained awake and transmits the data frame. However, simulations show that the adaptivity of the protocol is still very limited. T-MAC shuts down the radio too aggressively and introduces a high delay for multi-hop transmissions. The performance gain of the traffic adaptivity enhancement further only pays off for non-uniform bursty traffic patterns, since after the rather short timeouts have expired, all nodes are turning off their radios. Packets that are received from the upper layer in-between two T-MAC active periods, are scheduled for the next regular common active period.

X-MAC [25] is based on asynchronous wake-up intervals, during which nodes poll the channel for a preamble containing their address. *Buettner et al.* [25] derive a formula for finding the energetically optimal sleep-wake interval, given that the traffic is of Constant Bit Rate (CBR) with rate r that is unknown to the network in advance. The proposed algorithm finds the *near-optimal* wake-up interval given a traffic rate r, which is depends on the following relation: when increasing the

wake-up interval of the nodes, they consume less energy on average for polling the channel, *but* each frame transmission becomes more costly, since the expected number of strobes needed to reach a receiver increases. The traffic adaptation mechanism is, however, not implemented in Contiki's X-MAC implementation [170]. Since the basic medium access scheme of X-MAC using strobed preambles requires a certain minimal interval between two wake-ups, and because of the generally high per-packet overhead consisting in at least one strobe and Early-ACK before every data frame transmission, the maximum throughput still remains limited. It reaches only a fraction of that of energy-unconstrained wireless MAC protocols. Chapter 7 presents a solution that goes much beyond the maximum throughput of X-MAC, at the price of a temporally higher energy cost.

The **AMAC** [111] protocol relies on the S-MAC active window structure consisting in SYNC, RTS and CTS windows. With low traffic, AMAC neglects the costly RTS/CTS exchange and operates with a large sleep interval between two active periods. With increasing traffic, it multiplies the amount of active periods by a factor of 2^n, thus increasing the net duty-cycle by the same factor. Applying this adaptation strategy, the protocol increases throughput to some extent. With our contributions of Chapter 7, we follow a similar adaptation strategy of multiplying the active period frequency by factors of two, but even go a step further and let nodes temporarily abandon any sleep-wake pattern. However, with choosing the slotted S-MAC mechanism with a rather large static listen interval of 10% duty-cycle, AMAC's idle energy footprint remains considerably higher than that of many other more recent E^2-MAC protocols. Compared to state of the art preamble-sampling approaches exhibiting idle duty-cycles below 1% (c.f. Chapter 7), an idle duty cycle of 10% must clearly be considered inefficient.

Zero Configuration (ZeroCal) [124] is an optimization framework on top of the Contiki [47] X-MAC layer, which chooses the crucial X-MAC parameter settings (e.g., the wake-up interval) at run-time instead of compile-time, by calculating the near-optimal wake-up frequency based on the encountered traffic rate. The current settings are periodically recalculated and updated, but not communicated to the neighboring nodes, as for instance in our contributions in Chapter 7. Communicating the current settings would generally not make sense, since ZeroCal is based on X-MAC, which does not store and use information regarding the wake-up schedules of the neighboring nodes. By taking into account the consumed energy of the child nodes, parameters are set such that the energy consumption of the network is somewhat balanced. The adaptation strategy has been shown to increase the network lifetime by up to 50%, compared to static network-wide configuration.

Low Power Listening with Wake up after Transmissions (LWT-MAC) [28] is a traffic-aware MAC protocol that attempts to combine the benefits of slotted protocols and asynchronous random access protocols. Under low load conditions, the protocol employs the random access scheme of the B-MAC [143] protocol. With increasing load, a scheduled access scheme is is adopted right after a B-MAC transmission, which is similarly structured as that of the S-MAC protocol. With

low traffic load, B-MAC shows slightly better performance with respect to energy consumption and delay. With increasing traffic, however, LWT-MAC succeeds in increasing throughput and decreasing latency, since collisions of the long preambles significantly degrade B-MAC's performance.

In the **Burst-Aware Energy-Efficient Adaptive MAC (BEAM)** [12] protocol, the wake-up interval is adapted at run-time depending on the state of the packet queue. In the current implementation of BEAM within the Contiki OS, a node duty-cycles its radio with a wake-up frequency of 64 Hz at maximum, and 4 Hz at minimum. In contrast to our contributions in Chapter 7, the currently used interval of a node is not propagated to neighboring nodes. Similarly as done in ZeroCal, BEAM bases on the X-MAC channel access technique, which simply sends out strobes whenever a packet has to be sent without taking the neighboring nodes' wake-up schedules into account. A pairwise parameter exchange of the current wake-up frequency is hence rendered obsolete, for the same reasons as discussed with ZeroCal.

Wake-up Radio Protocols

The authors of the **Power Aware Multi-Access (PAMAS)** Protocol [166], the Berkeley **PicoRadio** [70] project and the **Sparse Topology and Energy Management (STEM)** [160] system have proposed to employ two types of radios for solving the E^2-MAC problems of a) idle listening and b) the need for internode time-synchronization at the moment of data transmission. Their employed node platform is equipped with the *main radio* for transmitting and receiving data frames, and the *wake-up radio*, which is an ultra low-power transceiver of lower complex circuitry. In PAMAS, the wake-up radio is used for exchanging RTS/CTS packets and then waking up the main radio for the data transmission and reception. In PicoRadio, the wake-up radio is assumed to be much simpler than in PAMAS or STEM, and is always kept turned on. When a node has pending packets, the wake-up radio transmits a simple signal that triggers the receiving stations to turn on their main radios. The experimental dual-channel chip PicoRadioRF, however, has been shown to be prone to interferences and noise in the channel. Often, nodes wake up their main radios when in fact they are not targeted receivers. In STEM, the wake-up radio is more complex and is capable to reliably encode the target destination address. Due to its higher complexity compared to PicoRadio, however, the designers of STEM have chosen to duty-cycle the wake-up radio, and propose to run a simple low-power-listening approach for this radio as well.

To the best of our knowledge, there has not been too much research activity in this class of protocols in the recent past. We conjecture that the research in this area has come to limits that seemed difficult or even impossible to overcome.

2.4.5 Trends & Future Directions

An enormous amount of work has been dedicated to designing and prototyping various wireless sensor MAC protocols in the past decade. A large number of pro-

tocols has been developed for very specific application areas and use cases, where they provide an added value compared to existing approaches, e.g., due to better performance with respect to energy-efficiency, throughput, reliability or latency. However, as research in the area of sensor networks progresses towards real-world use cases and the development of consumer-oriented commercial and industrial products, we are convinced that in the near future, fewer work will focus on isolated topics such as medium access control or routing. As in many technologies that in the end have to be applied by non-expert users in various other fields of research or in business models of the economy, there is a need for *standardization*.

In the past couple of years, we have observed the undeniable trend that in the majority of deployment studies, especially where the MAC layer itself was not of primary importance, random access protocols have made their way and have become the predominantly applied class of protocols. With X-MAC/ContikiMAC and B-MAC being the *default* MAC layers in Contiki and TinyOS, respectively, preamble-sampling MAC protocols are nowadays by far the most widely used protocols in deployment studies. Since these concrete implementations, after years of improvement and bugfixing, have proven robust and error-resilient, and further exhibit low idle duty-cycles, they nowadays qualify for a very wide range of real-world applications. Therefore, preamble-sampling MACs probably have become one of the standards that the field was looking for throughout the last decade.

Evidence in Recent Related Work

In the following, we briefly describe a selection of application studies of WSNs in real-world deployments, where the MAC layer issue itself does not play a central role. All of these studies have been published at the scientifically most reputable conferences in the sensor networks field, notably the *ACM International Conference on Embedded Networked Sensor Systems (SenSys)* or the *International Conference on Information Processing in Sensor Networks (IPSN)* and the *European Conference on Wireless Sensor Networks (EWSN)* between 2008 and 2010. We have observed that the researchers carrying out these projects have all chosen preamble-sampling protocols on the MAC layer for their software configurations - although, especially for TinyOS and Contiki, other MAC protocol implementations were also readily available.

- *Ceriotti et al.* [30] monitored the temperature and humidity conditions of medieval frescoes in the *Torre Aquila* deployment, as well as the resulting deformations of the tower. The deployment lasted for over 4 months. The authors use the TeenyLime framework on top of TinyOS' B-MAC implementation.
- The same authors have applied sensor network technology to adaptively control the electric light in a highway road tunnel in [29]. They used wireless sensors to sense the present natural light intensity at different locations within the tunnel to minimize the additional energy spent for artificial light. Again, the authors utilized the default TinyOS B-MAC implementation.

- *Pasztor et al.* [138] and *Dyo et al.* [54] have studied the social patterns of badgers during one year deployment of a wildlife monitoring system in a forest environment in Great Britain. Using TmoteSky nodes with Contiki and X-MAC, the authors reached an effective net duty-cycle of 1%, and could hence prolong the network's lifetime in order to meet the requirements of their study.
- *Chipara et al.* [35] describe the design, deployment and empirical analysis of a wireless clinical monitoring system that collects pulse and oxygen saturation readings from 41 patients in a step-down hospital unit over 7 months. The authors utilize TinyOS and its default B-MAC implementation. The end-to-end reporting reliability achieved in this study was reported higher than 99%.
- *Chang et al.* [32] empirically study the oxygen concentration and temperatures in differences depths of a small lake in Denmark using wireless sensor nodes attached to several buoys. The authors use TinyOS and its default B-MAC implementation.

This excerpt only serves to illustrate the undeniable trend towards application of preamble-sampling protocols in all kinds of application scenarios. A vast number of additional studies could undoubtedly be quickly identified in other venues. Unless researchers were interested in promoting their own proposed MAC protocols by applying it in a field study, almost no study of the past three years could be identified using a *frame-based* or *slotted* energy-efficient MAC protocol.

Why Asynchronous Preamble-Sampling?

We investigated the reasons why the class of asynchronous random-access and preamble-sampling MAC protocols seems to have made its way to become the most predominant class of MAC protocols in today's sensor network communities. We are certain that the following characteristics of this class of MAC protocols play a major role and are often decisive when it comes to design the operating system and network stack configuration for deployment studies:

- Asynchronous preamble-sampling protocols do generally not rely on assumptions and prerequisites, which might in reality turn out to be cumbersome to achieve. One such example is the rigid time-synchronization requirement of frame-based or slotted MAC protocols, which is not necessary in asynchronous random-access based protocols. MAC protocols with a continuous slot or frame assignment mechanism are prone to failure if the network connectivity within the network becomes (temporarily) disrupted. Intermittent connectivity is often encountered in real-world deployments, often because of a slight change in the environment, e.g., because of a new obstacle in between two nodes (like a closed door), or because some nodes move and hence get out of each other's range, e.g., in a wildlife-monitoring application. Further problems relate to assumptions concerning the clock drifts: synchronized MAC protocols rely on constant drifts and periodically re-synchronize. However, in practice, clock drifts can differ heavily from one node instance to another, since digital con-

trolled oscillators (DCOs) can become affected by variations in temperature, pressure or humidity conditions. Asynchronous protocols do not rely on the assumption of a network-wide time-synchronization, which qualifies them for scenarios where a temporal network partition can not be excluded.

- Preamble-sampling MAC protocols are general-purpose protocols, offering flexible *best-effort* communication. Their energy footprint in the idle traffic case is limited to a few channel checks. The per-packet overhead for contention and channel acquisition is furthermore comparatively low. For many scenarios, running these protocols hence result in the longest lifetimes, and the network service characteristics they provide often suffice as well, although they can not guarantee any minimal Quality of Service.
- Preamble-sampling MAC protocols are of low complexity, since no distributed slot assignment or scheduling problems have to be solved. This usually results in less complex and hence smaller code for the MAC layer. Since sensor node microcontrollers are heavily limited in memory (e.g., 64 KBytes for the popular MSP430 [171]), this can be a decisive advantage, as developers are often struggle to have enough memory left for their applications.
- Due to their general-purpose nature and good performance in many case studies, the preamble-sampling MAC protocols X-MAC/ContikiMAC and B-MAC have been chosen as *default* MAC layers in Contiki and TinyOS, respectively, although there were other prototype implementations of MAC protocols present in the development repositories. Since the scope of researchers working in the WSN field is more and more moving towards designing complex application-oriented systems with WSN technologies, and the observation of real-world phenomena, altering the MAC layer for slightly better in-situ performance is in most cases not considered. Hence, the choice of the *default* MAC layer in Contiki and TinyOS has probably set the direction for the standard of the next decade of WSN studies and for many real-world use cases and applications.

There is evidence that the basic scheme of asynchronous wake-ups and preamble-sampling will emerge as a standard for the MAC layer in many future studies, especially for WSN applications and deployment studies. The availability of reliably working implementations has the effect of a *de-facto* standardization, and might ultimately converge towards one MAC protocol solution. Just like IEEE 802.11 has competed versus HIPERLAN/2 by the earlier availability and thus higher market penetration to become the standard for wireless broadband LANs, we expect that, finally, some sort of asynchronous contention-based preamble-sampling MAC protocol will dominate the market of industry-ready WSN technologies. Therefore, with our contributions of in Chapter 7, we build upon the established design principles of asynchronous preamble-sampling, instead of *re-inventing the wheel* of MAC protocol design. However, we target at *alleviating* certain *drawbacks* of this class of MAC protocols, especially their weak adaptability to variable and high traffic volumes and the inherent limitation of throughput and latency compared to energy-unconstrained wireless channel MAC protocols.

2.5 Forward Error Correction

A key factor for the proliferation of wireless sensor network technologies is the reliability of the communication. It is crucial for many applications that the sensed data is delivered quickly and reliably across the network. The low-power wireless channel is, however, often prone to many hard-to-predict wireless phenomena. Transmission errors can occur due to a variety of reasons, ranging from multipath propagation effects, fading, scattering and reflection to interferences with other ongoing transmissions. Often, the channel exhibits a timely and spatially variable bit error rate (BERs) in the range of 10^{-4} or even higher, which results in packet loss rates ranging from less than 1% to sometimes far more than 10%-20%, as described by *Zhao et al.* [191]. In wired networks, BERs are usually several magnitudes lower (e.g., at maximum 10^{-7}, but usually in the range of 10^{-9} for DSL networks, as explored by *Starr et al.* [168]).

In this thesis, we tackle the issue of unreliable and lossy links and their performance-degrading impact on Quality of Service in several contexts, ranging from the MAC layer to the transport layer. The techniques presented in this section relate to our contributions on link-quality-aware run-time adaptive Forward Error Correction in Chapter 8. A few selected figures were taken by courtesy of *Sebastian Barthlomé* from his thesis presented in [17].

2.5.1 Automatic Repeat reQuest vs. Forward Error Correction

High error rates on the link level inevitably lead to a higher rate of corrupted packets, rendering the retrieved data unusable. The simplest and most naive way to deal with transmission errors on the link layer is to retransmit the same packet again until it has been correctly received or a maximum retry count has been reached. RFC 3366 [62] describes different *Automatic Repeat reQuest (ARQ)* schemes that are used today in different kinds of networks, ranging from various wireless networks to wireline and optical networks. In ARQ, the sender appends a cyclic redundancy checksum (CRC) [141] to the transmitted packet and waits for the acknowledgement (ACK) from the receiver. In order to reliably determine the integrity of the packet, the receiver calculates the CRC across the received payload again and compares it to the received checksum. If both CRCs match, the receiver confirms the successful reception to the sender with an ACK. If the sender does not receive an ACK within a certain time window, it assumes that the transmission attempt has failed and invokes a retransmission.

A sophisticated mechanism to cope with packet corruption due to random bit errors is the concept of *Forward Error Correction (FEC)*, as described by *Shannon* in [163]. FEC is used in a wide range of commercial and industrial products where data is transmitted over erroneous channels and where, henceforth, bit errors are likely to occur. FEC is based on *Error Correcting Codes (ECCs)*, which can detect and correct a certain amount of errors in a sequence of bits. In FEC, the sender computes parity information according to the applied ECC over the data bits and

adds this redundant information to the payload. At the receiver, the decoder of the applied ECC checks the received data bits for errors by taking this parity information into account. FEC schemes hence generally introduce an overhead with respect to computation (encoding/decoding) and the number of bits to be transmitted. However, this overhead can pay off with an increased packet delivery rate (PDR) and a reduction of the (re)transmission overhead and packet latency, since in case an error occurs, the correction can take place right after packet reception.

2.5.2 Error Correcting Codes

The Error Correcting Code (ECC) is the key component of any Forward Error Correction (FEC) mechanism. The literature basically distinguishes between two basic types of codes: *Block Codes* and *Convolutional Codes*. Block codes split the raw data to be encoded into a predetermined amount of bits (a block) and encode it block after block. Convolutional codes work on bit streams of arbitrary length, but are often converted into block codes in practice by defining a constant size block unit. Error correcting codes can *detect* and *correct* bit errors - in contrast to cyclic redundancy checksums (CRCs), which can only judge on the presence of such errors. The number of detectable and correctable errors differs heavily among the different ECCs. However, since the pure error detection capabilities of most ECC codes are much below that of CRC, FEC are usually combined with CRC schemes. FEC schemes can usually only detect one or a few more errors than they can correct. In contrast, CRCs can reliably determine whether a decoded payload is equal to the originally sent payload. The study [122] provides a detailed analysis of the *collision probability* of CRC16 and CRC32, hence the probability that two input sequences map to the same checksum, which equals to $\frac{1}{2^{16}} = 1.5 \cdot 10^{-5}$ and $\frac{1}{2^{32}} = 2.3 \cdot 10^{-10}$ for CRC16 and CRC32, respectively. Even with CRC16, it is hence very unlikely that bit errors introduced into the payload is left undiscovered, as it would have to map to the same checksum as the correct payload. CRC16 is the 16-bit variant of CRC, which is applied in many MAC protocol implementations for WSNs, including those of this thesis. The CRC16 implementation we used is based on a pre-calculated lookup table with 256 entries stored in the text segment, which significantly speeds up the calculation of a CRC across a packet payload.

An important characteristic of the different ECCs is the way how the encoded information is represented. In a *systematic code*, the original unencoded bit sequence is visible in an unchanged form within the encoded bit sequence. Systematic codes have a decisive advantage over non-systematic codes: given that no bit errors have occurred (which can be verified using a CRC), the decoding process is simple: the original information can be decoded by just removing the parity information from the received information. In *non-systematic codes*, the original data is not visible as clear text in the encoded data. Hence, these codes always require decoding operations, which can be time-consuming and costly, depending on the complexity of the employed code. In the following, we briefly characterize four classes of Error Correcting Codes, upon which we rely our further investigations in Chapter 8.

Repetition Code

The Repetition Code [127] is probably the most simple and straightforward way to achieve the capability to correct bit errors. It requires very few computational power, however, at a price of a high parity overhead in the resulting encoded data.

Encoding: The encoding operation of the Repetition Code takes k bits from the raw data and simply repeats them r times, with r being an odd number ≥ 3. When implementing the repetition code, we chose to operate on the byte level, hence taking units of 8 bits as input. The Repetition Code's encoding function then simply consists in copying each input byte three times to obtain the encoded data.

Decoding, Error Detection and Correction: The error detection and correction procedure in the Repetition Code is straightforward. The parameters k and r used for encoding define how many bits are used to encode one single bit. Hence, each of the 0-bits and 1-bits of the encoded bits are counted one after the other. The decoder then decides the value of the *decoded bit* according to the higher counter. With r being an odd number, it is guaranteed that one of the counters will have a higher value than the other. The decoding hence relies on the assumption that the majority of the encoded bits have not been flipped - a mechanism which is often referred-to as *Majority Logic Decoding* [184]. The error correction capability of the repetition code is limited to $\lfloor \frac{r}{2} \rfloor$, i.e., a repetition code with $k = 1$ and $r = 3$ can correct 1 bit error in every 3 bits of the encoded data.

Hamming Code

The Hamming Code is probably the most prominent and well-known ECC and belongs to the *systematic* linear block codes. In a hamming code, the bit length of an encoded word n is computed as $n = 2^m - 1$, where $m = n - k$ equals the number of parity bits, which explains the commonly used notation of Hamming(n,k).

The calculations of the Hamming Code heavily rely on matrix operations, involving the two matrices *Generator matrix* $\mathbf{G}$ and the *Parity check matrix* $\mathbf{H}$, which have the following form:

$$\mathbf{G}_{k,n} := (\boldsymbol{I}_k | - \boldsymbol{A}^T) = \left[\begin{array}{cccc|cccc} i_{11} & & & & -a_{11} & -a_{21} & \cdots & -a_{n-k1} \\ & i_{22} & & & -a_{12} & -a_{22} & \cdots & -a_{n-k2} \\ & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ & & & i_{kk} & -a_{1k} & -a_{2k} & \cdots & -a_{n-kk} \end{array}\right]$$

$$\mathbf{H}_{n-k,n} := (\boldsymbol{A} | \boldsymbol{I}_{n-k}) = \left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1k} & i_{11} & & & \\ a_{21} & a_{22} & \cdots & a_{2k} & & i_{22} & & \\ \vdots & \vdots & \ddots & \vdots & & & \ddots & \\ a_{n-k1} & a_{n-k2} & \cdots & a_{n-kk} & & & & i_{n-kn-k} \end{array}\right]$$

The matrix $\mathbf{A}$ is contained in both the generator matrix $\mathbf{G}$ and the parity check matrix $\mathbf{H}$, and is composed of the parity bits calculated over the data bits, cf. *Ling*

	d_0	d_1	d_2	d_3
p_0	1	1	0	1
p_1	1	0	1	1
p_2	0	1	1	1

Figure 2.17: Relation of Data and Parity Bits

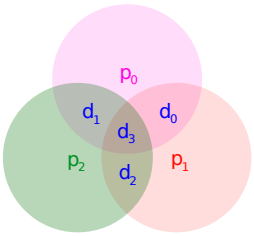


Figure 2.18: Data-Parity Bit Coverage

et al. [116]. The basic idea of the Hamming Code is to calculate the parity bits in such a way that each original data bit is covered with a different unique set of parity bits. Since all calculations are performed on binary numbers, all the matrix operations are computed in the Galois Field GF(2) using modulus 2. More details on the construction of $\mathbf{A}$ and $\mathbf{H}$ and their properties can be found in the textbooks of *Ling et al.* [116] and *Moon* [126].

Encoding: Based on the chapter on Hamming codes in [126], we give a brief example of Hamming(7,4) and construct $\mathbf{A}$ from the parity bits calculated from a block size of 4 raw input bits. The encoding uses the Generator matrix $\mathbf{G}_{4,7}$, which has the following form:

$$\mathbf{G}_{4,7} := \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The generator matrix $\mathbf{G}_{4,7}$ contains the identity matrix $\mathbf{I}_4$ in the first four columns. The subsequent three columns correspond to the transposed inverted matrix $-\mathbf{A}^T$. Table 2.17 lists the data bits $d_0, d_1, ..., d_3$ in the columns and the three parity bits p_0, p_1, p_2 in the rows. Each parity bit in each row can hence be calculated or expressed as the sum of the listed data bits in the Galois Field GF(2). Note that in GF(2), the expression 1 + 1 equals to 0, which can easily be achieved with the XOR instruction. As the rows are linearly independent, each data bit d_i can be represented as the sum of a unique set of parity bits.

Figure 2.18 illustrates this relationship: the data bit d_0 can be calculated from the set of parity bits $\{p_0, p_1\}$ and the data bit d_3 from the parity bits $\{p_0, p_1, p_2\}$. Any input bit sequence expressed as the vector $\boldsymbol{u} = u_0 u_1 ... u_{k-1}$ can hence be encoded as the matrix multiplication $\boldsymbol{v} = \boldsymbol{u}\mathbf{G}$, which results in a sequence of n bits.

Decoding, Error Detection and Correction: The so-called minimum Hamming distance d_{min} of the employed code determines its error correction and detection capabilities. d_{min} is the minimum number of bits that have to be flipped in order to transform one valid code word into another one, or the minimum distance between any two code words of the Hamming Code. The error correcting capability t can be calculated as follows:

$$t = \lfloor \frac{d_{min} - 1}{2} \rfloor$$

In the example of the Hamming(7,4) code, at least three bits have to be changed to obtain a new code word, hence $d_{min} = 3$. Hamming(7,4) hence is able to correct 1 bit error per code word. The parity check matrix $\mathbf{H}$ for the Hamming(7,4) has the following form, cf. [126]:

$$\mathbf{H}_{3,7} := \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

In the Hamming code, encoding and decoding can likewise be modeled as vector addition operations and matrix multiplications. The received encoded word $\boldsymbol{r}$ can be written as the sum of the two vectors $\boldsymbol{e} + \boldsymbol{v}$, where the vector $\boldsymbol{e}$ models the bit error positions introduced to the encoded code word $\boldsymbol{v}$, hence $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$.

The so-called syndrome vector $\boldsymbol{s}$, can be computed from the received data $\boldsymbol{r}$, the transposed parity check matrix $\mathbf{H^T}$, and the relation $\boldsymbol{v} = \boldsymbol{u}\mathbf{G}$:

$$\boldsymbol{s} = \boldsymbol{r}\mathbf{H^T} = (\boldsymbol{v} + \boldsymbol{e})\mathbf{H^T} = \boldsymbol{v}\mathbf{H^T} + \boldsymbol{e}\mathbf{H^T} = \boldsymbol{u}\mathbf{G}\mathbf{H^T} + \boldsymbol{e}\mathbf{H^T} = \boldsymbol{e}\mathbf{H^T}$$

since it can be shown that the following condition holds, cf. [126]:

$$\mathbf{H}\mathbf{G}^T = \mathbf{G}\mathbf{H}^T = 0$$

The syndrome vector $\boldsymbol{s}$ contains information with which the location of potentially introduced bit errors can be derived. If all values of the vector are equal to zero, no bit error occurred. In this case, no error has to be corrected. In case the syndrome is not zero, errors need to be corrected. The bits which are equal to one indicate which bits were flipped and need to be corrected.

Double Error Correction Triple Error Detection Code

The Double Error Correction Triple Error Detection (DECTED) proposed by *Gulliver et al.* [69] is a *systematic* block code from the class of the Hamming codes. The DECTED(16,8) variant used in this thesis is able to correct up to two bit errors and detect three adjacent bit errors. DECTED(16,8) encodes 8 input data bits and creates an encoded word of 16 bits, of which the first 8 bits correspond to the original input bits and the second 8 bits are the parity bits.

Encoding: The encoding and decoding process is analogous to that of the Hamming(7,4) code and consists in an encoding operation using a generator matrix $\mathbf{G}$ and decoding using the parity check matrix $\mathbf{H}$ to obtain the syndrome vector. The DECTED(16,8) encoder takes 8 bits and computes the encoded word matrix using $\mathbf{G}_{8,16}$, which has the following form, cf. [69]:

$$\mathbf{G}_{8,16} := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Encoding a byte $\boldsymbol{u}$ to obtain a 2-bytes code word $\boldsymbol{v}$ corresponds to a multiplication of $\boldsymbol{u}$ with the generator matrix, hence $\boldsymbol{v} = \boldsymbol{u}\mathbf{G}$.

Decoding, Error Detection and Correction: In the case of DECTED(16,8), the matrix $\mathbf{H}$ has the following form:

$$\mathbf{H}_{8,16} := \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In analogy to Hamming(7,4), the syndrome vector $\boldsymbol{s}$ is calculated using the parity check matrix $\mathbf{H}$ and the received code word $\boldsymbol{r} = \boldsymbol{e} + \boldsymbol{v}$, and the relation $\boldsymbol{v} = \boldsymbol{u}\mathbf{G}$:

$$\boldsymbol{s} = \boldsymbol{r}\mathbf{H}^{\mathbf{T}} = (\boldsymbol{v} + \boldsymbol{e})\mathbf{H}^{\mathbf{T}} = \boldsymbol{v}\mathbf{H}^{\mathbf{T}} + \boldsymbol{e}\mathbf{H}^{\mathbf{T}} = \boldsymbol{u}\mathbf{G}\mathbf{H}^{\mathbf{T}} + \boldsymbol{e}\mathbf{H}^{\mathbf{T}} = \boldsymbol{e}\mathbf{H}^{\mathbf{T}}$$

The syndrome vector $\boldsymbol{s}$ then again contains the location of the erroneous bits. If all numbers within $\boldsymbol{s}$ are zero, there is no error to correct, and the decoder just needs to chop off the green parity bits from the corrected code word. In case there is a non zero value, $\boldsymbol{s}$ uniquely identifies a 1-bit error or a 2-bit error within the received encoded data $\boldsymbol{r}$. A more detailed description concerning the detection and correction of bit errors in DECTED(16,8) can be found in [69].

Bose-Hocquenghem-Chaudhuri (BCH) Code

Bose-Hocquenghem-Chaudhuri codes were invented in 1959 by *Hocquenghem* [79] and independently in 1960 by *Bose* and *Ray-Chaudhuri* [22] and have since then been intensively studied in academic research, and likewise been adopted in many commercial and industrial applications, e.g. in CD-ROM drives, where read errors often occur due to scratches or dust on the disk, or in the Digital Video Broadcasting Terrestrial (DVB-T) standard, where errors must be recoverable by the receiver alone because of the unidirectional channel, which does not allow the receiver to

request a retransmission or signal a reception failure by not sending an acknowledgement. BCH codes belong to the class of *systematic cyclic* block codes, which have convenient algebraic properties that qualify them to be extensively used in the context of error correction coding. BCH codes are highly flexible and can be tailored to correct any number of bit errors in a code word of a given length. BCH are defined by the two parameters n and k. A BCH(n,k) code encodes k bits into a code word of length n bits.

Encoding: The essential component for encoding in BCH is the *generator polynomial* $g(x)$, with which the data is processed in a block-by-block manner, similar to the generator matrix mechanism in the Hamming code. In the BCH code, encoded bit sequences correspond to polynomials in the Galois Field (GF). Each code word can be represented as the sum of products of the generator polynomial with a data block. *Morelos-Zaragoza* [127] and *Lin* [115] provide a detailed mathematical discussion of the generator polynomial. BCH relies on the operation of the modular division of polynomials. A sequence of input bits u is first transformed into the polynomial $u(x)$. The encoded polynomial $v(x)$ is then calculated as

$$v(x) = u(x) \cdot x^k - ((u(x) \cdot x^k) \mod g(x))$$

where $u(x) \cdot x^k$ is a so-called *cyclic shift* of the original bit sequence of k positions, with k representing the degree of the generator polynomial. Since the original input is shifted by the k bits to the higher positions, it remains visible in clear text within the code word, which qualifies BCH as a *systematic code*.

Decoding, Error Detection and Correction: In analogy to the Hamming code, which calculates a syndrome vector, BCH computes a so-called *syndrome polynomial* $s(x)$ that locates the positions of the errors, if there are any. An encoded code word $r(x)$ can be written as

$$r(x) = v(x) + e(x)$$

with $e(x)$ being the polynomial representing the occurred errors. The syndrome polynomial $s(x)$ is computed as

$$s(x) = r(x) \mod g(x) = e(x) \mod g(x)$$

If the resulting syndrome polynomial equals 0, hence $s(x) = 0$, no errors occurred and hence there is no error to correct. Since BCH is a systematic code, the decoding then just consists in retrieving the bits that represent the original input data. In case of a non zero value of the syndrome, the BCH correction mechanism comes into play: First, the erroneous bits in the received code word are determined. Two algorithms have been proposed for this task, among them is the Berlekamp-Massey algorithm [18][120]. Berlekamp-Massey calculates the unknown error polynomial $e(x)$ out of the syndrome polynomial $s(x)$. When $e(x)$ is known, the so-called *unique roots* of the error polynomial have to be found to determine the error positions using the Chien Search algorithm [34]. A detailed mathematical discussion of the BCH decoding and error correction process can be found in *Hong et al.* [81].

Reed-Solomon (RS) Code

Reed-Solomon Codes have been proposed in 1960 by *Reed and Solomon* [154]. They belong to the class of cyclic systematic block codes, and share many similarities with BCH codes. An RS code specified as RS(n,k) encodes k bits into a code word of n bits. Just as BCH codes, RS codes be easily parametrized to correct a specified amount of errors, they have been applied in various industrial and consumer products and a wide range of networking technologies, e.g., DSL, WiMAX, the Advanced Television Systems Committee (ATSC) standards, as well as in commonly used data storage technologies. The error correcting capability t of any given RS(n, k) code calculates as $t = \lfloor \frac{n-k}{2} \rfloor$.

Encoding: In Reed-Solomon, the original data is interpreted as coefficients of a polynomial $p(x)$ over the Galois Field GF(2^m), where m specifies the amount of bits denoting the symbol size. Just as in BCH, a generator polynomial $g(x)$ for the given RS(n, k) is used for the encoding operation. $g(x)$ has the following form:

$$g(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3 + \ldots + g_{2t-1} x^{2t-1} + g_{2t} x^{2t}$$

where $2t = n-k$ equals the number of parity symbols. The variable t denotes how many errors the code can correct per block, and $2t$ (=the degree of the polynomial) equals the number of parity bits per block. The coefficients $g_0, g_1, \ldots, g_{2t}$ are determined by n and k. The encoded data equals the coefficient of the output polynomial $s(x)$ that is obtained when multiplying $p(x)$ with $g(x)$ in GF(2^m):

$$s(x) = p(x) \cdot (x)$$

Decoding, Error Detection and Correction: The received code word that can have been corrupted during transmission is again interpreted as the received polynomial $r(x)$, for which the following holds:

$$r(x) = s(x) + e(x)$$

where $e(x)$ denotes the error location polynomial. For determining $e(x)$, several algorithms can be used, among them again the Berlekamp-Massey [18]. The so-called *unique roots* of the error polynomial $e(x)$ then again determine the error positions, and can be determined using the Chien Search algorithm [34]. More details on the Reed-Solomon encoding and decoding can be found in *Moon* [126].

2.5.3 Forward Error Correction in Wireless Sensor Networks

A recent analysis of related work on Forward Error Correction mechanisms and run-time adaptive schemes in particular applied in WSNs, conveyed that the related work in this field is yet rather limited. Most studies have applied basic FEC techniques and simple ECCs in the context of sensor and mobile ad hoc networks. However, the chosen codes and the applied experiment methodologies clearly leave room for further investigations. Run-time adaptive FEC schemes have yet only been studied in network simulators and in rather different networking contexts

(e.g., IEEE 802.11-based MANETs), other than static WSNs with low-power narrowband channels.

Jeong et al. [99] study the performance of two rather simple ECCs. The study evaluated Hamming-based SECTED and DECTED codes in an indoor and outdoor link scenario using the Chipcon CC1000 [172] radio in the frequency range around 868 MHz and simple on-off-keyed modulation (OOK). However, the entire analysis was conducted in the absence of any other ongoing concurrent traffic. The authors of [99] conveyed that the most frequently occurring errors in their experiment were 1-bit and 2-bit errors, and that occasionally, a few burst errors occurred in their indoor experiment. The authors conclude that complex ECCs with a high correctional power are hence not necessarily required, but that this trade-off needs to be investigated in environments with higher error rates.

Busse et al. [27] similarly examine Hamming-based SECTED and DECTED codes and a Reed-Solomon variant in an indoor testbed of the FU Berlin Embedded Sensor (ESB) nodes, which use the RFM TR 1001 [156] radio chip that operates with the same frequency and modulation as the CC1000. Reed-Solomon are briefly discussed in Section 2.5.2. The ESB nodes, which we also used in Chapter 5 for a preliminary study on MAC layer mechanisms, are further described in Section 2.1.1 of this chapter. Unfortunately, the topology and the traffic pattern examined is far from real-world environmental conditions: 16 nodes are placed in a corridor in a line-of-sight with a distance of only 1m in between two nodes and a rather large payload size of 255 bytes. The impact of signal distortions caused by the signal penetrating concrete walls and floors is hence not taken into account. Similarly as in [99], the effect of interferences from concurrent transmissions, which have a crucial impact in real-world sensor networks, were neglected as well. In [99], only one node is broadcasting packets, which are then logged by the remaining 15 nodes. The authors of [27] conclude that most errors were 1-bit and 2-bit errors, but that burst errors occurred rather frequently. Henceforth, they propose to apply code word *interleaving* as an effective strategy to cope with burst errors.

Willig et al. [185] also use the ESB platform and evaluate a very similar setup, namely a chain of 10 nodes with a distance between two adjacent nodes fixed to only 30 cm. However, the study is limited to evaluate the frequency and the occurrence patterns of transmission failures due to non-detection of the frame preambles or because of the occurrence of bit errors. Once again, important crucial wireless phenomena were not taken into account, e.g., signal attenuation through concrete walls and floors or interfering transmissions of other nodes in the network.

Liang et al. [114] is the most recent study on ECCs in WSNs. It appeared during our own evaluations on ECC mechanisms, cf. Chapter 8. The study examines one Hamming and one Reed-Solomon variant to protect transmissions of a TelosB network from interferences with an IEEE 802.11b/g wireless LAN, which operates in the same 2.4 GHz ISM band. [114] shows that indeed, the packet delivery rates could be improved by up to 70% using ECC, especially during phases of heavy interferences caused by conference attendants using their IEEE 802.11 devices.

2.5.4 Adaptive Forward Error Correction

Since Forward Error Correction (FEC) schemes are able to correct bit errors without making an entire retransmission necessary, they could potentially be employed in WSNs, which are traditionally susceptible to high bit error rates and weak signal-to-noise ratios. However, since their application comes at the cost of an increased power consumption due to complex encoding and decoding operations, their application is only justified when there is a real necessity. This basically motivates the need for run-time adaptive FEC schemes, which adapt the level of error coding to the encountered link quality. To the best of our knowledge, extensive real-world experiences with adaptive FEC schemes applied in distributed low-power WSNs do not exist to date. In Chapter 8, we bridge this missing gap with our thorough evaluation of eight different ECCs and three adaptive ECC selection strategies in a series of real-world experiments.

Ahn et al.[1] presents simulation-based study on adaptive FEC mechanisms. This study examines three static BCH configurations and an adaptive approach, the so called FEC-level adaptation (FECA) algorithm. In FECA, the parameters n and k of BCH are adapted in three steps, in order to increase and decrease the amount of parity bits depending on the channel conditions. FEC adaptations are activated by either a packet loss or the timeout of a backoff timer. However, FECA [1] is not particularly designed for sensor networks, but rather for IEEE 802.11-based wireless mobile ad hoc networks. The authors use the network simulator ns-2 [133] and apply a generic wireless channel error model. Their study concludes that FECA performs better than the application of statically configured FEC mechanisms, given that the error rates do not oscillate too rapidly.

Ahn, Hong et al. [2] introduce the adaptation algorithm AFECCC in their network simulation-based study. This algorithm dynamically matches the FEC code size to the low-frequency wireless channel bit error rate, which is assumed to be measurable by the sensor node. According to various simulations with different channel models, AFECCC performs better than any static FEC setting.

In the recent past, the application of simulation tools have been identified as a general drawback of many ad hoc and sensor network studies. Inappropriate parameter settings, unrealistic channel, traffic and error models of many simulation studies have led to an erosion in trust in simulation results, cf. *Kurkowski et al.* [105] and *Andel et al.* [11]. The trend in research on wireless sensor and ad hoc networks has clearly shifted towards experimental feasibility studies of proposed mechanisms and protocols on real-world devices, cf. [95]. Therefore, our contribution in Chapter 8 clearly distinguishes from the related work presented above. Instead of simulating ECCs with a user-defined wireless channel error model and interpreting questionable simulation result data, the chapter evaluates eight different ECCs, ranging from simple codes with a low correctional power to complex and sophisticated codes with a high error correction capability in a wide range of experiments performed under real-world sensor network conditions.

2.6 TCP/IP in Wireless (Sensor) Networks

In this thesis, we tackle the problem of TCP/IP-based communication within WSNs operating with radio duty-cycling energy-efficient MAC (E^2-MAC) protocols. This section presents works and studies that are related to our contributions in Chapter 9. Selected figures were taken by courtesy of *Ulrich Bürgi* from his thesis [26].

2.6.1 Problems of TCP in Wireless Networks

TCP/IP in wireless ad hoc networks has long been observed to perform poorly, especially with increasing network size and communication across multiple hops. Reasons for this vital handicap have been intensively studied in the context of IEEE 802.11-based ad hoc networks [31][80][68], long before sensor networks emerged. The main reasons for poor TCP performance in wireless multi-hop networks can be summarized as follows:

- In wired networks, packet loss could be safely associated to congestion in the network. With TCP developed over years or even decades for communication in wired networks, most TCP variants (e.g., RENO, SACK) significantly reduce the congestion window when perceiving packet loss. The so-called AIMD (Additive-Increase/Multiplicative-Decrease) algorithm lets the congestion window grow linearly, but reduces it exponentially when a congestive situation is perceived. In RENO, the sender reduces the size of its congestion window to half when a packet loss is detected. Together with the *fast recovery* and *fast retransmit* mechanism described in RFC 2581 [6], this congestion control strategy has proven highly effective in various wired networks.
- The wireless channel is prone to transmission errors caused by multipath propagation, short-lived interferences, fading, scattering and reflection effects and other wireless phenomena. Often, links in WSNs exhibit a timely and spatially variable bit error rate (BER) in the range of 10^{-4} or even more, which results in packet loss rates ranging from less than 1% to sometimes far more than 10%-20%, as shown by *Zhao et al.* in [191]. In wired networks, where TCP/IP was initially designed for, BERs are several magnitudes lower (e.g., DSL networks [168] exhibit BERs of maximum 10^{-7}, but usually in the range of 10^{-9}). The default TCP congestion control mechanism interprets packet loss as a sign of congestion in the network, and aggressively throttles the congestion window (multiplicative decrease), which has a vastly deteriorating impact on the medium utilization and consequently on the end-to-end throughput. In wireless networks, packet loss often occurs *at random*, e.g. because of short-lived interference or reflection effects, and not necessarily due to network congestion.
- Wireless networks applying contention-based random access MAC protocols can only make collisions (and hence TCP packet losses) less probable, but never fully prevent them. The reasons for this are manifold. First, nodes only have a half-duplex channel: they can not *receive* and *transmit* at the same time. Therefore, two nodes might concurrently sense the carrier idle and then start

to transmit, without noticing the collision they caused. Second, the assumption of *symmetric links* (if A is in the range of B, B can also reach A) does often not hold. Collisions can therefore occur due to nodes interfering with transmissions out of their range. Third, contention-based protocols access the channel at random, often with low initial backoff windows (e.g., 7 slots in IEEE 802.11), which renders collisions in the initial transmission attempts rather likely.

- In wireless networks, the so-called *hidden node* and *exposed node* problems often lead to collisions of TCP data segments and, even more often, TCP acknowledgements. Since data segments are much larger, ACKs are more likely to collide with them, which results in a higher share of missing ACK segments at the TCP sender. This in turn often lets the senders' TCP timeout timer expire, which then implies restarting the congestion window from the lowest level.

Most work on the topic of TCP in wireless multi-hop networks has been conducted in IEEE 802.11-based ad-hoc networks. A frequently cited study is Snoop [15] (cf. Section 2.6.2), upon which many further studies on TCP over multiple wireless hops rely. Snoop is, to the best of our knowledge, the first study to introduce TCP caching and local retransmission techniques in intermediate nodes in a TCP connection, which is the key mechanism of our contribution in Chapter 9. We neglect to discuss other IEEE 802.11-based studies in this chapter, since in general, the wireless channel and device constraints in IEEE 802.11-based networks are rather different from those of sensor networks (i.e., channel bandwidth, computational power, memory and energy consumption constraints). Studies that explicitly tackle TCP optimizations in WSNs have been rather rare. We briefly discuss the most well-known studies *Distributed TCP Caching (DTC)* [46] and *TCP Support for Sensor Networks (TSS)* [24] in Section 2.6.3 and Section 2.6.4, respectively.

2.6.2 Snoop

Snoop [15] addresses the TPC/IP performance in IEEE 802.11-based local area networks with high packet loss. One endpoint of the examined scenario is a mobile notebook computer connected to a WLAN access point over one wireless link. The access point itself then communicates across an Ethernet-based local area network with the second endpoint. The study hence tackles the most frequently used scenario of IEEE 802.11-based devices, namely the case where only the *last hop* from, e.g., a notebook to the access point is wireless. Figure 2.19 illustrates this scenario.

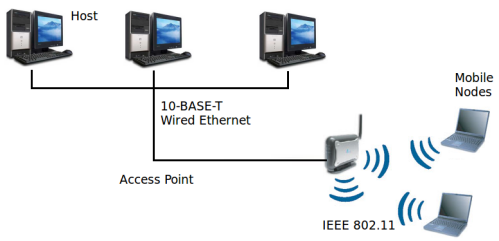


Figure 2.19: Experiment Scenario in the Snoop Study [15]

In Snoop [15], the access point continuously monitors TCP connections and caches all segments sent from the wired network to the notebook, until the respective TCP acknowledgements (TCP ACK) are retrieved. If a packet loss is detected on the wireless link, either due to a timeout or due to duplicate ACKs, the access point retransmits the affected segments and drops duplicate ACKs that occur in the meantime. Therefore, the endpoint in the wired network is kept unaware of any transmission errors on the wireless link to a large extent, since it would interpret these packet losses as a result of network congestion. By letting the access point inspect the IP packets and intervene on the transport layer - hence beyond the layer it is traditionally working in - Snoop significantly improves the end-to-end throughput by preventing that the TCP congestion control mechanisms are triggered.

Further features of Snoop are the support of Selective Acknowledgements (SACK), which were not yet widely supported in TCP at the time Snoop was developed, and which have since then been integrated into TCP with RFC2018 [121]. When the access point detects a packet loss, it immediately replies with a SACK for requesting the missing segments, which reduces the amount of end-to-end retransmissions. Furthermore, access points running Snoop forge SACKs when they do not receive any packets from a mobile device for a certain amount of time. In case the mobile device had sent packets to the access point that have not been correctly received, periodically generated SACK packets from the access point reduce the waiting time for the retransmission.

2.6.3 Distributed TCP Caching for Wireless Sensor Networks

Distributed TCP Caching for Wireless Sensor Networks (DTC) [46] is the first contribution proposing TCP assistance mechanisms for TCP in sensor networks. DTC proposes to cache TCP segments on forwarding sensor nodes in order to perform hop-by-hop retransmissions and minimize end-to-end retransmissions, with the goal to increase end-to-end throughput, reduce latency and increase energy-efficiency. The mechanism can be seen as a generalization of Snoop [15] across multiple links, since Snoop only targeted at the single wireless link between an WLAN access-point and a mobile wireless node, e.g., a notebook computer.

DTC is illustrated Figure 2.20, with one sender (cf. *TCP Client*), three intermediate nodes A,B,C and one receiver. Each intermediate node forwarding packets of a TCP connection maintains a cache entry for buffering one TCP segment. When receiving and forwarding a TCP segment, each node caches it with a probability of 50%, given that the cache entry is not already full. A node participating in the TCP connection checks the acknowledgment number of an incoming TCP ACK with the sequence number of its cached segment. If the number is greater than the sequence number of the cached segment, the cached segment is removed and the TCP ACK is forwarded towards the *sender* node, cf. (a) in Figure 2.20. An acknowledgment number smaller than or equal to the cached segment's sequence number indicates a packet loss: in case of an equal number, the node simply retransmits the cached segment, cf. Figure 2.20 (b). In case of a smaller number, DTC makes use of

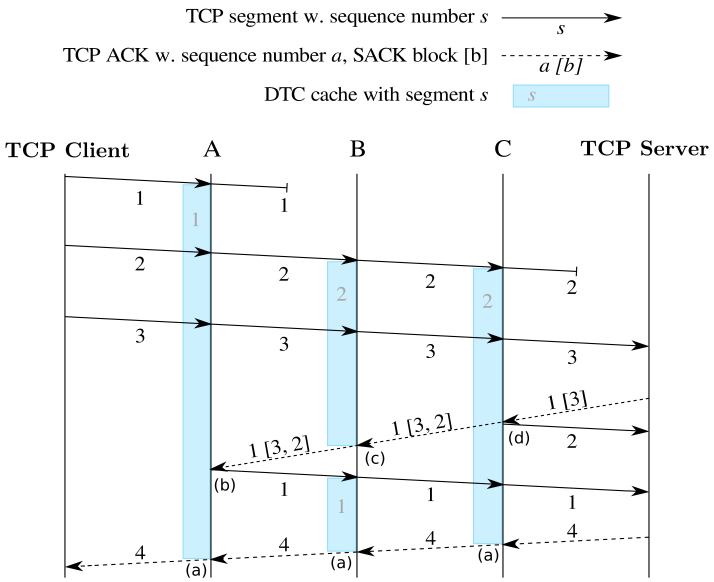


Figure 2.20: Distributed TCP Caching and Local Retransmissions

TCP's Selective Acknowledgment (SACK) option. In case the SACK block of an incoming TCP ACK packet already contains the sequence number of the cached segment, the node receiving the ACK knows that another node in-line with the *receiver* node has cached the same segment as well, and thus empties its cache and forwards the ACK towards the *sender* node, cf. Figure 2.20 (c). In case the number of the cached segment is not already in the SACK block, the node receiving the ACK retransmits its cached segment towards the receiver, adds the sequence number of its cached segment to the SACK block and forwards this ACK towards the sender, cf. Figure 2.20 (d). Besides observing TCP ACK packets, DTC also makes use of MAC-layer ACKs: if a node does not receive a MAC-layer ACK for a TCP segment it forwards, it schedules a timer for a retransmission.

The evaluation using OMNeT++ [178] in the DTC study [46] has shown that in a scenario with 6 hops and a wireless channel of 10% packet loss, DTC significantly reduced the amount of transmissions and increased the throughput by 450%.

2.6.4 TCP Support for Sensor Networks

TCP Support for Sensor Networks (TSS) [24] is an extension of Snoop [15] and DTC [46] tailored for use in resource and especially energy-constrained sensor networks applying TCP over multiple wireless hops. Similar to DTC [46], intermediate nodes within a TCP connection cache TCP segments to trigger local retransmissions. After transmitting a segment, it is kept cached until the node can be sure that the next node in line has correctly received it. This can be accomplished by receiving a TCP acknowledgement for the segment, or by overhearing the next node transmitting the segment further on. This behavior makes sure that a congestive situation is resolved quickly, since when one node stops forwarding

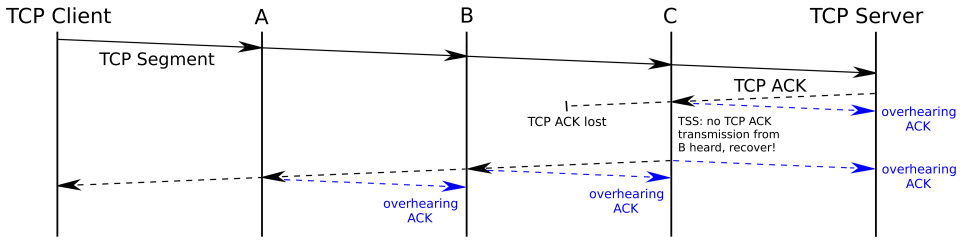


Figure 2.21: Local ACK Recovery in TCP Support for Sensor Networks (TSS)

segments because of a packet loss on the next link in line, the preceding nodes will stop transmitting segments as well.

Each node participating in the TCP connection continuously updates an estimation RTT_{est} of the round-trip time from the node itself to the TCP endpoints of each TCP connection, using the original TCP Exponentially Weighted Moving Average (EWMA) filter discussed in RFC 793 [147]. If after a TCP segment transmission, the respective TCP acknowledgment has not arrived after $1.5 \times RTT_{est}$, the segment is regenerated from the cache, and a local retransmission is initiated. TSS furthermore proposes local regeneration of TCP acknowledgments: An incoming TCP segment that has already been acknowledged is not forwarded, but answered with a locally generated TCP acknowledgement with the highest acknowledgement number the node has ever received. In addition, TSS introduces the so-called *Aggressive TCP Acknowledgement Recovery* mechanism, which consists in overhearing the next node's transmission of a previously forwarded TCP acknowledgement. If the time between the own transmission of the TCP acknowledgement and the forwarding transmission of the latter by the next node is larger than 2 times its average value, the TCP acknowledgement is sent again, as illustrated in Figure 2.21, where node C's transmission of the TCP acknowledgement is not received by B, and hence regenerated and sent again by C.

TSS was evaluated in OMNeT++ [178] and has been shown to significantly reduce the total amount of transmissions required to forward 500 TCP segments over a chain of 10 nodes (-70%). The simulation settings assumed that nodes are configured with an energy-unconstrained CSMA/CA MAC layer, which permits the overhearing of the transmissions of the next nodes in line. In the presence of radio duty-cycling E^2-MAC protocols, however, some TSS features would not be generally applicable. Features that can only be accomplished when each node's radio transceiver is kept turned on continuously, or at least after each frame transmission, would require substantial modifications on the MAC protocol. However, such modifications would most likely negatively impact on the overall energy-efficiency as well. Due to this lack of real-world experiences with DTC [46] and TSS [24], we study the performance of their basic concepts with several energy-unconstrained *and* radio duty-cycling E^2-MAC protocols in Chapter 9 in a series of real-world experiments on our testbed facilities.

Part I

Frameworks and Tools

Chapter 3

Wireless Sensor Network Testbed Design and Management

In this chapter, we present our management framework for WSN testbeds, the *Testbed Management Architecture for Wireless Sensor Networks (TARWIS)*. The TARWIS system is one of the core contributions resulting from the involvement of the Research Group on Computer Networks and Distributed Systems of the University of Bern in the European-Union WISEBED [162] project, which targeted at establishing an experimental federation of testbeds of WSNs, all of them made accessible over the Internet to the European WSN research community.

TARWIS has been used as a key tool in the major part of experimental evaluations throughout this thesis and many other experimental studies that are yet to appear, i.e. the studies [88][96][94]. TARWIS fundamentally improved our methodology of controlled and repeatable experimentation in real-world WSN testbeds, which is the reason why we discuss its key features and innovations in detail in this chapter. Section 3.1 discusses the main rationale behind the development of TARWIS. Section 3.2 then illustrates the design of TARWIS and its core components, the workflow of using TARWIS to reserve testbed resources, schedule and execute experiments on a WSN testbed, as well as the real-time experiment monitoring capabilities. Section 3.3 discusses the data representation standards on which TARWIS relies for the testbed description and the output of the results. Section 3.4 discusses the testbed facilities located at the University of Bern Engehalde Campus, which was set up during the course of the WISEBED project, and which was used throughout several chapters of this thesis. Section 3.5 concludes the chapter.

3.1 Motivation

For years, simulation has been the research tool of choice in the majority of wireless ad-hoc and sensor networks studies. However, with discovering wide gaps between simulation results and real-world results of many distributed algorithms, e.g., MAC and routing protocols, the appropriateness of simulation tools for simulating wireless phenomena has more and more been been questioned. Especially in the

sensor and ad-hoc network community, inappropriate parameter settings and unrealistic radio, traffic and/or mobility models have been identified and criticized as a general drawback of simulation studies, cf. the studies of *Kurkowski et al.* [105] and *Andel et al.* [11]. With research in this field growing more mature, researchers have generally aimed at proofing the real-world feasibility of their proposed protocols and mechanisms on real-world devices. For the purpose of evaluating protocol behavior in practice, experimental sensor network testbeds have become indispensable in the WSN field today.

In the past five years, numerous universities and research institutions have started to set up real-world sensor network testbeds. In most cases, these testbeds have been set up for research and teaching purposes, in order to enable testing and evaluation of real-world behavior of developed protocol mechanisms. An increasing number of stationary WSN testbeds have been put into operation, with different node hardware and very heavily differing architectural testbed design. The most prominent examples are Harvard University's MoteLab [182], the TWIST testbed [75] of TU Berlin or the Kansei [60] testbed of Ohio State University. The testbed management software solutions implemented for these testbeds have, however, generally been tightly coupled to one particular testbed deployment, and are hence not easily reusable for further testbed setups. Still today, researchers setting up an own testbed of WSNs are often starting from scratch to implement testbed management features, which are as simple as user account management, experiment resource reservation, configuration and scheduling, or a consistent representation of results.

We intend to bridge this gap with the TARWIS management architecture presented in this chapter. In contrast to almost all of the software employed for the testbeds discussed in Section 2.2 of Chapter 2, TARWIS [92] has been kept independent of the underlying testbed organization or the sensor node hardware and software.

3.2 Testbed Management Architecture TARWIS

In this section, we pinpoint the main advantages of TARWIS over existing testbed management solutions for wireless sensor network testbeds, before we continue to describe TARWIS in a more detailed manner in the subsequent sections.

- While most testbed management solutions have been implemented around a specific testbed architecture and node type, TARWIS has been kept as independent as possible of most crucial design questions of its underlying testbed hard- and software. TARWIS can hence be used to control and manage testbeds with a single server architecture (to which nodes are connected, e.g., over USB cables), with a two- or three-tiered architecture where gateway mesh nodes control the access to the individual mesh nodes, or to testbeds where no wired control channel is present at all. TARWIS only requires *that* the sensor nodes of the testbed can be remotely controlled and accessed from within a central location, the so-called *portal server*. TARWIS makes no restrictions *how* this is actually achieved. The TARWIS components communicate over standardized

and programming-language independent APIs, which keeps the reusability and generality of TARWIS on a high level.

- Most of the existing testbed management solutions interact with specific features of the sensor node and its operating system (in most cases TinyOS [112]), or intermediate gateway nodes, e.g., the popular NSLU2 devices [117] running an embedded Linux [134], and thereby hence lose their generality. TARWIS is totally independent from the sensor node type and sensor node operating system. Besides our testbed at University of Bern, it has been deployed on eight other testbeds throughout Europe, using at least five different node types and operating systems. The universities of Lancaster, Lübeck, Braunschweig, Berlin, Delft, Geneva, Patras and UPC Catalunya can all be accessed with TARWIS, forming the pan-European testbed federation of WISEBED [162].
- Unlike in most other testbed management solutions, where experiments are usually set up and run invisibly in batch-mode, TARWIS allows the testbed user for monitoring and interacting with the ongoing experiment at run-time by observing the output of the selected sensor nodes in a browser window. TARWIS offers the same technical capabilities to the experimenting testbed user as if the sensor nodes would be attached to its desktop computer: nodes can be *reprogrammed*, *reconfigured* or *hard-reset* over the browser window at experiment run-time. The TARWIS monitoring screen displays each node's output in a small dynamically reloaded HTML textbox, which permits the experimenting user to observe the output of an entire network on one screen.
- TARWIS has been designed to integrate multiple testbeds from different universities or research institutions into the Shibboleth Federation [164] of the WISEBED [162] project. TARWIS offers an integrated and federal approach for user authentication, authorization and account management, and relieves the testbed operators from designing user management solutions from scratch. Each user account of the Shibboleth federation can be used for all testbeds that are accessible over a TARWIS deployment, and for institutions deciding to join the Shibboleth federation in the future, rendering a per-site registration obsolete. The federation approach massively reduces the overhead of bookkeeping the federation users, which impacts directly on the administrative overhead to maintain the testbeds operational and available.
- TARWIS integrates the proposed Wireless Sensor Network Markup Language (WiseML) [40], an XML standard schema for describing experimental data in WSNs. In an attempt to achieve compatibility of sensor network experimental data with several simulation tools, the WiseML standard has been adopted by the well-known COOJA [59] simulator in *Li et al.* [113]. WiseML is a cornerstone towards a unified representation of experimental data in the field, may it be from experiments on simulators or real-world data traces. TARWIS is the first architecturally generic and fully WiseML-compatible testbed management and experiment monitoring system. It uses WiseML for parsing the network topology, and for representing the results after experiment completion.

3.2.1 TARWIS Architecture

The architecture of TARWIS is illustrated in Figure 3.1. The figure displays the portal server, on which the essential parts of TARWIS are hosted. Besides the TARWIS server component (lower left corner), the portal server hosts the TARWIS web interface (within an Apache web server), which is protected by the authentication and authorization system Shibboleth [164]. In this section, we briefly describe the different components visible in Figure 3.1 along with the technologies applied to implement and realize them.

Portal Server

The portal server can be any desktop PC with some minimum of 2 GB of RAM and a broadband Internet connection. TARWIS comes with installation scripts to simplify the setup, which have been optimized for Debian Linux [39] (v.5.0.0). Besides the portal server, a Shibboleth *Identity Provider* needs to be prepared and configured to interoperate with the WISEBED Where-Are-You-From (WAYF) server located at University of Bern, which manages the authentication mechanisms.

TARWIS Web Interface

TARWIS offers an intuitive and easy-to-use web-based user interface. This interface is implemented as a dynamic web page using PHP [142], offering the user convenient access to the testbed via a web browser. A MySQL [131] Database Management System (DBMS) is used to store personal configurations and experiment definition data. In order to gain access to the TARWIS web interface, a user has to log in using its Shibboleth credentials. Based on the user identification credentials, TARWIS offers fine-grained access to the testbed resources - either as *visitor* (only being able to observe public-declared experiments), as a normal *testbed user* (being allowed to submit experiment jobs) or as a *testbed administrator* (a root-like account that, besides submitting experiments, is also capable of adding nodes, schedule maintenance tasks, delete normal users' experiments, etc.).

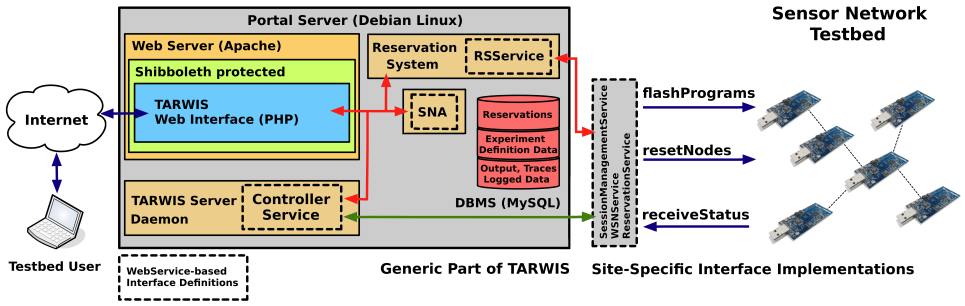


Figure 3.1: TARWIS System Architecture Testbed Generic Part (left) and Testbed Specific Implementation (right)

Federated Authentication and Authorization Infrastructure

The TARWIS architecture separates the concerns of authentication and authorization to the resources. Authentication of users logging in to the portals is based on Shibboleth [164]. When signing in to any TARWIS deployment, a user has to enter its user name and password at the so-called *home organization*. Each user's home organization is responsible for authenticating its affiliated users.

Shibboleth is a standards-based, open source authentication and authorization system used for Single-Sign On (SSO) and user/account management across organizational boundaries. To date, it is widely adopted in European education and research networks (e.g., SWITCH [123], DFN [42], eduGAIN [55]). The computer science libraries IEEE Xplore [97] and Elsevier ScienceDirect [58] support Shibboleth in order to facilitate access from universities worldwide.

TARWIS Server Daemon

The TARWIS Server Daemon is the main process controlling the entire experiment execution logic. It encapsulates the database access and all the logic operations defined in the WSDL description files. The TARWIS Server Daemon forks a designated subprocess as soon as any new experiment is scheduled. This subprocess then fetches the experiment description from the database, and checks with the TARWIS Reservation System whether the submitted experiment has a valid reservation. It determines which nodes have to be reprogrammed with which binary code image, then connects to the sensor nodes in order to reset and reprogram them. Then, the TARWIS Server Daemon retrieves the run-time experiment data from the sensor nodes, which is subsequently stored in the database.

It is important to point out that *every* interaction between TARWIS and the testbed, e.g., the services for reprogramming or retrieving output, are based on well-defined *Web Service* calls. This makes sure that TARWIS remains *generic* and *reusable*.

Web Services-based APIs/Interfaces

The Web Services standard [175] offers an unambiguous and machine-processable notion to define software component interfaces, the so-called *Web Services Description Language (WSDL)*. Web Services interface descriptions are independent from the programming language or operating system. There are Web Services bindings for any major programming language on all major operating system platforms, and they are more and more applied in business and industry today.

Figure 3.1 displays three examples of such API functions, which are invoked by TARWIS: reprogramming a node using the *flashPrograms*, resetting nodes using *resetNodes*, and retrieving output or status messages from a node with *receiveStatus*. The implementation of these services can reside on the same computer as the TARWIS components, or any other computer that is accessible with a high bandwidth connection from the portal server machine.

Resource Reservation System

The TARWIS Reservation System is used to prevent concurrent access to the resources on the testbed, hence to guarantee uninterruptible experimentation of multiple users on the testbed. Reservations can be made for the entire testbed, or for subsets of testbed resources, with a subset consisting of at least one node. The TARWIS Reservation System exposes its primitives to the TARWIS Web Interface, where it can be manipulated in a browser window. Apart from that, it can also be queried and manipulated in a machine-driven manner (e.g., using a script calling the reservation-related Web Service). A fully interoperable reservation client, which communicates over the same WSDL functions, is furthermore available as iPhone application, permitting to *book* timeslots from within a simple smartphone.

The ReservationSystem API consists in primitives to obtain a list of the current reservation (e.g. *getReservations*, *getConfidentialReservations*) and to schedule or manipulate reservations (e.g., *makeReservation*, *deleteReservation*).

Sensor Network Authentication Service (SNA)

Each TARWIS deployment further exhibits a Web Service for authorization. The process of authorization - granting rights to authenticated users - is done locally for each TARWIS testbed. Each testbed operating university can specify which access right is granted for which specific user from any home organization in the WISEBED federation. To ensure modularity and consistency, the so-called Sensor Network Authorization Service (SNA) is hence present as a Web Service on the Portal Server. It can additionally be queried by any other Web Services enabled client, e.g., a Perl or Python script.

3.2.2 Experimentation using TARWIS

This section illustrates the workflow of using TARWIS to schedule, configure and run an experiment on any TARWIS-administered testbed. The subsequent screenshots have been made at the TARWIS deployment of University of Bern. The TARWIS Web Interface has four main tabs for actions related to *Reservation*, *Experiment Configuration*, *Experiment Monitoring*, and *Testbed Management*, as depicted in the upper half in Figure 3.2. Depending on the role of the user in the testbed (visitor, user, administrator), access rights are defined such that some tabs are accessible and some are not (e.g., the testbed management tab is accessible for administrators, but not for users or visitors).

Reservation

The *Reservation* tab offers a user interface for querying and manipulating the Web Services-based TARWIS Resource Reservation system. The main reservation overview screen is depicted in Figure 3.2. The screen per default depicts the

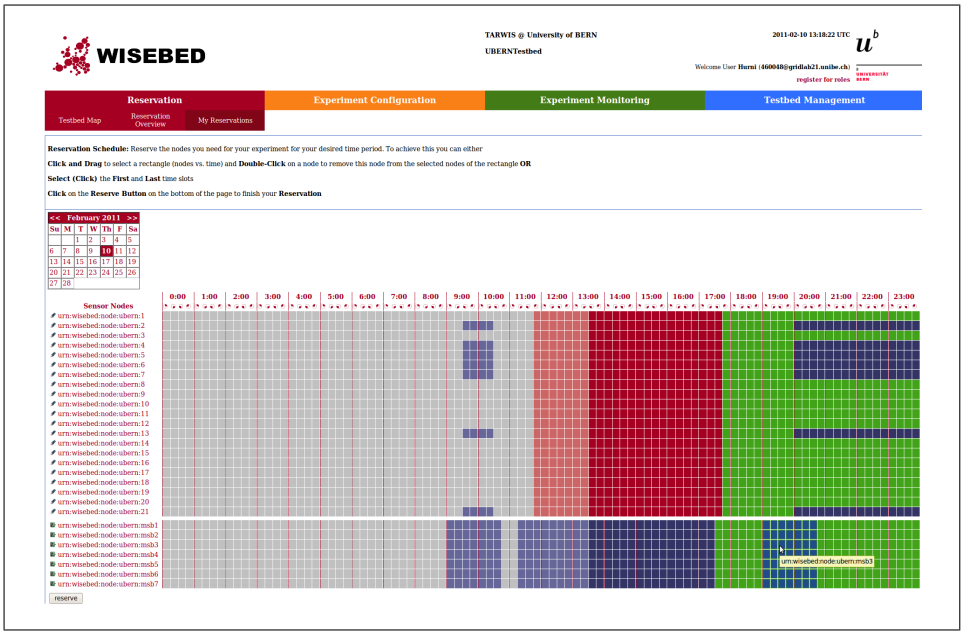


Figure 3.2: TARWIS Reservation Screen

current day, but experiments can be scheduled at any time in the future. The screen lists the node resources, sorted by the node types on the y-axis, versus the time on the x-axis. The time is divided into indivisible units of 15 minutes.

Figure 3.2 depicts five scheduled reservations, of which two are already past, and two are ongoing. One reservation in the lower right corner is about to be submitted. The experiments scheduled by the current user are colored in dark blue. The user taking the screenshot obviously had scheduled two reservations on a subset of 7 TmoteSky/TelosB nodes (listed on the left), and two reservations on the MSB430 [14] sensor nodes. The free and unallocated time slots are colored green, whereas the time that is already past and that has not been allocated is colored grey. Past or partly-past reservations are colored with a grey shade, as clearly visible in Figure 3.2 before 12.30 UTC. Clicking on a rectangle and dragging with the mouse cursor selects a rectangle of nodes and time units, which define the time and the affiliated resources of that particular reservation. The 7 MSB430 [14] nodes are just about to be selected for a reservation starting at 19.00 UTC, as displayed in the bottom right of the figure. After pressing the *reserve* button on the bottom of the page, the selected resources are reserved for the subscribed user.

Experiment Configuration using TARWIS

The third tab in the TARWIS Web Interface implements the *Experiment Configuration* process. All user-specific experiment and configuration data can be manipulated in this tab. In the first sub-tab, the user can store sensor node code images, i.e., those binary images, which are usually compiled with the mspgcc toolchain [129]

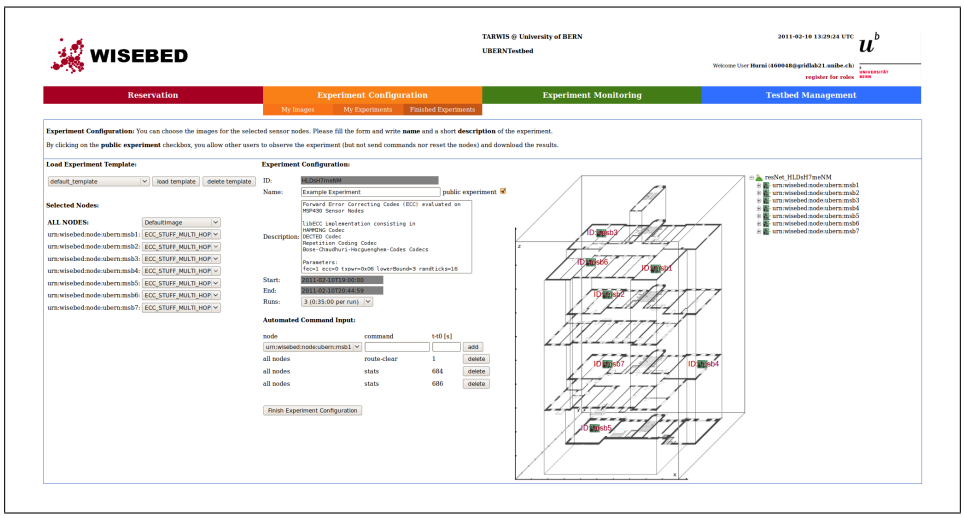


Figure 3.3: TARWIS Experiment Configuration Screen

for the majority of the sensor node platforms. The images are uploaded to the database with a user-supplied name and a unique identifier.

The second sub-tab of the *Experiment Configuration* tab relates to the configuration (or modification) of the scheduled reservations of the visiting TARWIS user. Figure 3.3 depicts the screen where the user can define the experiment configuration for the reservation he/she scheduled beforehand. On the left, each sensor node has to be assigned a sensor node code image. One can assign one image to all nodes, or configure nodes individually, assigning different images for each node.

In the middle of the screen, the experiment can be given a name and a brief description in the designated input and text fields. With the *public* checkbox, the user can let other users observe his experiment at run-time, and also permit other users to later download the experiment results. The node map on the right depicts the selected nodes and their position within the testbed.

Experiment Runs

Experimental research is often driven by the need to obtain statistically significant data. Researchers hence usually have to carry out the same experiments multiple times in a row, preferably under the same or at least comparable conditions, in order to obtain the statistically significant amount of data that is required for a sound analysis. In order to simplify this procedure, TARWIS integrates the *Runs* option in order to define how many *experiment runs* the experiment shall consist of. The entire duration of the reservation is then split into this specified number of runs. After each run, the output is written to a WiseML file and added to a zip archive, which is then made available for download and sent to the user via email. Given that the experiment had been declared *public* in the configuration tab, the results are also accessible for download for all other users of the federation.

Automated Commands

On the bottom of the page just below the experiment description box and the *Runs* option, the user may add so-called *automated commands*. These commands are issued after a specified number of seconds after the flashing operation finished. The target destination of these *automated commands* can be specified in the dropdown-box. A user may broadcast a command to all the sensor nodes in his selected set of nodes, or select only one unicast command recipient. The option is particularly useful in case an experiment needs to be started synchronously, e.g., by sending a *start* command to all nodes at the same time in order to initiate an experiment with a particular algorithm that relies on a certain synchronicity.

Experiment Configuration Templates

Entering the configuration data for an experiment in TARWIS can become exhaustive and time-consuming, especially if there are many nodes that need to be reprogrammed with different code images, if all the input text fields have to be edited and if many automated commands need to be scheduled. Practice has shown that these steps can become tedious and repetitive, especially when many different but similar experiments need to be scheduled. In TARWIS, the user is hence given the opportunity to save the experiment settings as a *template*. When later scheduling another similar experiment, he/she may just select the previously saved template and reload it with the *load template* button. The previously saved experiment configuration is then loaded automatically (experiment description, run settings, public settings, automated commands, assignment of images to the nodes, etc.) and the user can just modify the changing parameters to reflect the intended settings of the new experiment.

My Experiments/Finished Experiments

When the user has finished configuring the experiments, he/she presses the *Finish Experiment Configuration* button. In the *My Experiments* sub-tab, he/she may still edit its pending experiment configurations given that the experiment has not yet started. The results of the finished experiments are available for download in the *Finished Experiments* tab.

3.2.3 Experiment Monitoring

The *Experiment Monitoring* tab offers to monitor running experiments at run-time. It is definitely one of the most crucial advantages of TARWIS over other testbed management systems, which usually run experiments in batch mode without giving the user the opportunity to monitor or even interact with them. Figure 3.4 depicts an excerpt of a running experiment in the *Experiment Monitoring* tab. When different experiments are running on different subsets of the testbeds, the user can choose

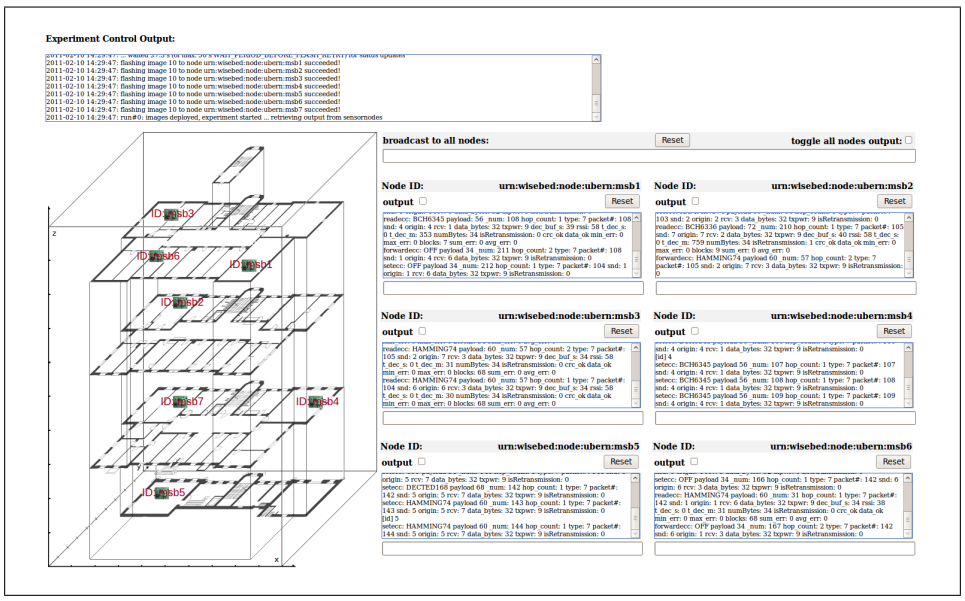


Figure 3.4: TARWIS Experiment Monitoring Screen

the experiment to be observed by clicking on a link displaying its ID, and switch between different experiments arbitrarily.

In the left part of Figure 3.4, an illustration of the testbed with the selected nodes positioned in the building of the testbed is displayed. On the top of the figure, the user can see the so-called *Experiment Controller Output* log. This text field logs the status messages of the TARWIS Server Daemon experiment-dedicated subprocess, which emits certain status information, e.g., at what time the nodes where reprogrammed (*flashed*), whether the operation was successful, which run is currently being executed, and much more.

On the right of the figure, the user can monitor the output of the selected sensor nodes. The six displayed output windows list the last 20 lines of each node, which were captured from the sensor nodes' serial interfaces. This output information is particularly useful for debugging sensor node applications. Our practice has shown that with observing the output windows from the individual sensor nodes concurrently at run-time, erroneous and unintended behavior and possible reasons for the latter can be much easier identified when observing the experiment in real-time, compared to tedious offline trace analysis in log files.

Below the output windows of the nodes, the user can enter commands to the sensor nodes, which are subsequently sent to the sensor nodes via the backend implementation and written to their serial interfaces. The user can hence interact with the nodes and, e.g., obtain status information using the TARWIS web interface in order to get a clearer picture why the examined protocol is not behaving as expected.

3.3 Experiment Results Representation in TARWIS

As soon as an experiment has been successfully scheduled and configured within the TARWIS Web Interface, its definition is stored in designated MySQL tables. When the scheduled experiment time is reached, the TARWIS Server Daemon sets up a Web Service designated to retrieve the experiment output via the *receive* primitive (cf. Figure 3.1) and subsequently stores this output in the database. When the experiment expires, all the retrieved output is exported to a file adhering to the Wireless Sensor Network Markup Language (WiseML). This WiseML file comprises all the significant information about an experiment, e.g., where the experiment took place geographically, what kind of nodes were used, what their configuration was, and most important the timestamped output of the nodes.

Using the Wireless Sensor Network Markup Language (WiseML) [40] standard for the representation of the results further allows for making the experiment data public to other research partners in a common well-defined language, giving them the opportunity to repeat the same or similar experiment and, e.g., trying to improve the results. WiseML is a cornerstone towards a unified representation of experimental data in the WSN field, may it be from experiments on simulators or real-world WSN testbeds. Three network simulators have to-date adopted the WiseML standard: the popular Contiki simulator COOJA [59], as well as the simulators Shawn [64] and WSNGE [101]. In addition, the mobility scenario generation and analysis tool bonnmotion [13] further supports WiseML. Integrating WiseML into the TARWIS testbed management system pushes experimental research on wireless sensor network testbed within the WISEBED testbed federation one crucial step forward towards a transparent and repeatable notion of the setup and scenario of sensor network experiments and their results.

Listing 3.5 shows an excerpt from a TARWIS-generated experiment trace in a small experiment at the University of Bern testbed. The WiseML format (as specified in [40]) first denotes general information about the testbed in the *setup* section, such as the experiment time and duration, the testbed name (here: UBERNTestbed), an unambiguous notion for the affiliated nodes and other details. In the *trace* section, the file lists the node IDs and the captured raw output, along with timestamps relative to the experiment start time. The timestamps are given in fractional seconds relative to the experiment start, with the precision depending on the time granularity the testbed supports (e.g., seconds, milliseconds, or microseconds).

The WiseML-file generated by TARWIS describes with a high accuracy what has happened at a particular time during the experiment. The experimenting user only needs to make sure that he/she is supplying enough information to TARWIS. Supplying information is as easy as using *printf*-like output on the serial interface throughout the experiment time - all these statements will finally be captured and integrated into the final WiseML experiment definition file.

```
<wiseml version="1.0" xmlns="http://wisebed.eu/ns/wiseml/1.0">
<setup>
 <timeinfo>
  <start>2011-02-10T19:00:00Z</start><end>2011-02-10T20:44:59Z</end>
  <unit>seconds</unit>
 </timeinfo>
 <description>UBERNTestbed</description>
 <node id="urn:wisebed:node:ubern:msb1">
  <position> <x>75</x> <y>30</y> <z>80</z> </position>
  <nodeType>MSB430</nodeType>
  <description>Server Room 3rd Floor</description>
 </node>
 <node id="urn:wisebed:node:ubern:msb2">
  <position> <x>20</x> <y>90</y> <z>52</z> </position>
  <nodeType>MSB430</nodeType>
  <description>FKI Room 203 (Indermuehle)</description>
 </node>
 <node id="urn:wisebed:node:ubern:msb3">
  <position> <x>12</x> <y>85</y> <z>80</z> </position>
  <nodeType>MSB430</nodeType>
  <description>RVS Pool 3rd Floor</description>
 </node>
 <node id="urn:wisebed:node:ubern:msb4">
  <position> <x>42</x> <y>90</y> <z>22</z> </position>
  <nodeType>MSB430</nodeType>
  <description>CGG Pool 1st Floor</description>
 </node>
 <node id="urn:wisebed:node:ubern:msb5">
  <position> <x>10</x> <y>70</y> <z>0</z> </position>
  <nodeType>MSB430</nodeType>
  <description>-111</description>
 </node>
  <node id="urn:wisebed:node:ubern:msb6">
  <position> <x>25</x> <y>50</y> <z>80</z> </position>
  <nodeType>MSB430</nodeType>
  <description>Node 3 - Student Pool 1</description>
 </node>
</setup>
<trace id="experiment_FEC_output"> [...]
 <timestamp>814.013436</timestamp>
 <node id="urn:wisebed:node:ubern:msb5">
   <data key="text">setecc: DECTED168 payload 68 _num: 142 hop_count: 1
   type: 7 packet#: 142 snd: 5 origin: 5 rcv: 7 data_bytes: 32 txpwr: 9
   isRetransmission: 0 </data>
 </node>
 <timestamp>814.26796</timestamp>
 <node id="urn:wisebed:node:ubern:msb7">
   <data key="text">readecc: DECTED168 payload: 68 _num: 142 hop_count: 1
   type: 7 packet#: 142 snd: 5 origin: 5 rcv: 7 data_bytes: 32 txpwr: 9
   dec_buf_s: 34 rssi: 38 t_dec_s: 0 t_dec_m: 65 numBytes: 34
   isRetransmission: 0 crc_ok data_ok min_err: 0 max_err: 0 blocks: 34
   sum_err: 0 avg_err: 0 </data>
 </node>
 <timestamp>814.390311</timestamp>
 <node id="urn:wisebed:node:ubern:msb7">
   <data key="text">forwardecc: BCH6336 payload 72 _num: 22 hop_count: 2
   type: 7 packet#: 142 snd: 7 origin: 5 rcv: 2 data_bytes: 32 txpwr: 9
   isRetransmission: 0</data>
 </node>  [...]
</trace>
</wiseml>
```

Listing 3.5: Excerpt from a TARWIS-generated Experiment WiseML-Trace

3.4 University of Bern Testbed

This section briefly portrays testbed facilities that were installed in the context of the European-Union WISEBED [162] project, and which were used for a major part of the evaluations conducted within this thesis. The evaluations described in Chapters 7, 8 and 9 mainly base on results of experiments which were conducted using the TARWIS testbed management framework.

The University of Bern testbed of wireless sensor nodes is located in the two buildings of the Institute of Computer Science and Mathematics at Neubrückstrasse 10 and Schützenmattstrasse 14 in 3012 Bern, Switzerland. The testbed to date consists of the following types and quantities of sensor nodes:

- 40 TelosB [144] nodes from Crossbow Corporation
- 7 MSB430 [14] nodes from Freie Universität Berlin & ScatterWeb GmbH

The University of Bern testbed hence to date consists of 47 sensor nodes of two different types. The testbed spans across the four floors of the building Neubrückstrasse 10 and two floors in the building Schützenmattstrasse 14. Figure 3.11 depicts the 47 nodes embedded into a 3-dimensional skeleton of the two buildings. The 7 MSB430 nodes are placed indoors in different rooms. Out of the 40 TelosB nodes, 39 nodes are placed indoors, each of them in different rooms or corridors of the building, as depicted in Figures 3.6 and 3.7 with two examples. One node is placed outdoor on the windowsill of the building's small tower, as depicted in Figure 3.8.

Ethernet-based Backbone Network

All sensor nodes are connected to the local area Ethernet network of the university campus by means of small mesh nodes or barebone PCs. The TelosB nodes are attached to mesh nodes of type WRAP from PCEngines [140]. A board of the employed type is depicted in Figure 3.9. These mesh nodes are equipped with a 233MHz AMD Geode SC1100 CPU, 128 MB SDRAM, 4GB Compact FlashCard and an IEEE 802.3 Ethernet interface. We have developed a generic build-system and a small Linux flavor [169] tailored for use in memory-constrained wireless mesh nodes. The result is a multi-purpose image that looks nearly like a standard

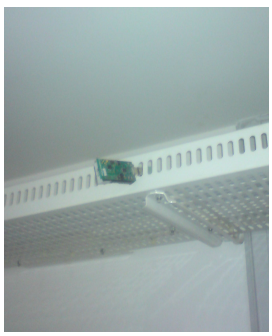


Fig. 3.6: Corridor

Fig. 3.7: Server Room

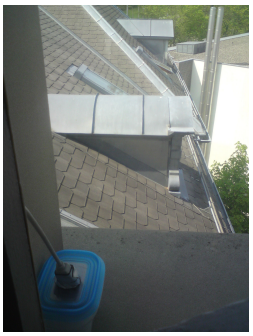


Fig. 3.8: Outdoor

Fig. 3.9: ALIX WRAP mesh node

Fig. 3.10: Asus EEPC

Linux system. The resulting image uses less than 20.0 MB in uncompressed form in RAM and compressed uses less than 5 MB on the flash device. All functionality to access the TelosB nodes was implemented top of the small Linux flavor. The MSB430 sensor nodes are attached to small inexpensive and i386-compatible desktop computers of the type Asus EEPC, depicted in Figure 3.10. The MSB430 nodes need to be reprogrammed with a tailormade JTAG-flasher Device from Texas Instruments, and can be accessed over the serial interface with an USB cable.

Both platforms, the ALIX mesh nodes and the Asus EEPCs, implement the following functionalities for the TelosB and MSB430 sensor nodes, respectively:

- Receiving and executing high-level commands over TCP/IP sockets to trigger actions such as *rebooting* the attached sensor nodes or *flashing* them with the binary image which is also received over the socket.
- Synchronization with the University of Bern NTP server (time.unibe.ch).
- Reading output data from the serial interface of the sensor node, sending it together with the current timestamp (in µs precision) to the TARWIS server.

Other than the ALIX mesh nodes, where most of the functionality is implemented in C daemons (except for the python-based bootstrap loader of the MSP430), the EEPCs are operated with a 32-bit Windows XP (SP3) and most functionality is implemented in Perl.

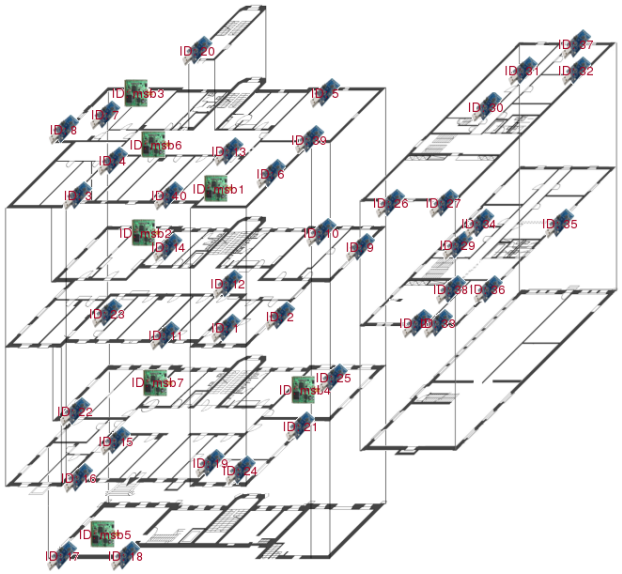


Fig. 3.11: The University of Bern WISEBED [162] Wireless Sensor Network Testbed

3.5 Conclusions

In this chapter we have described TARWIS, the *Testbed Management Architecture for Wireless Sensor Network Testbeds*. We have conducted the major part of the experiments described in this thesis using TARWIS, notably the experimental evaluations of Chapters 7, 8 and 9.

TARWIS is a *Web Services*-based generic management system for the administration and management of research testbeds of wireless sensor networks. TARWIS to date runs on nine different testbeds of the WISEBED [162] project, with node deployments between a few 10 to more than 100 nodes. The TARWIS management framework has been designed for federating testbeds of wireless sensor networks. Using Shibboleth, users can access any testbed within the WISEBED federation by using the same account credentials. The generic TARWIS web-based user interface, the standardized and programming-language independent Web Service interfaces of the TARWIS backend, as well as the node and testbed-architecture independent design of TARWIS permit interested research groups to make use of the framework for their own projected testbed. With the WISEBED project ending in 2011, TARWIS has been made publicly available on the University of Bern's Research Group on Computer Networks and Distributed System's website [83].

Our practical experiences have shown that using TARWIS, researchers working in the field of WSNs have a powerful instrument for prototyping and evaluating various sensor network protocols and mechanisms. The option to monitor experiments and interact with the entirety of the sensor nodes at run-time using a web browser is a powerful and unique feature of TARWIS and has not existed in any of the reviewed testbed management solutions before. The availability of the system notably expedited experimentation with our real-world distributed WSN testbed facilities, and due to its integration with WiseML can be seen as a contribution towards a controlled, repeatable experimentation methodology.

In this chapter, we have further described the University of Bern testbed that has been set up throughout the course of the European-Union WISEBED [162] project, and which was used for major parts of the thesis. The testbed to date consists of 47 sensor nodes, which are accessible over the campuswide Ethernet backbone. Using TARWIS, the testbed can be easily accessed from anywhere in the world using a simple web-browser. The WISEBED testbeds are to date regularly used by students as well as international researchers. Research results from these testbeds have yet been used for numerous scientific experiments, of which some have resulted in scientific publications, or are under submission, cf. [94][96][88].

Chapter 4

Software-based Energy-Estimation

In order to examine energy-conserving communication protocol mechanisms in the field of wireless sensor networks, which is the subordinate topic that spans across several chapters in Part II of this thesis, a simple, robust and reasonably accurate methodology to assess the energy consumption of a sensor node is an important cornerstone. Throughout this thesis, we have employed different methodologies to *measure* or *estimate* the energy consumption of sensor nodes. The methodologies to measure the energy consumption of small embedded devices as well as the physics behind them are described in detail in Section 2.3 of Chapter 2.

This chapter describes our own contributions to the research topic of *software-based energy estimation*, which is a technique that has become widely applied in the recent past. Section 4.1 motivates the need for reliable, robust and reasonably accurate software-based energy-estimation mechanisms in large-scale experiments of WSNs. Section 4.2 elaborates on the advantages and drawbacks of software-based energy estimation mechanisms and relates them to physical hardware-based energy measurements. Section 4.3 discusses our proposed software-based energy-estimation methodology, which was developed, refined and empirically verified in our study published in [86]. The section thoroughly elaborates on the model specification, model enhancement and the experimental validation of the accuracy of our software-based energy estimation methodology. The evaluations of the MaxMAC protocol prototype in Chapter 7 rely in part on the work presented in this chapter. With Section 4.4, the chapter concludes and gives an outlook on current and future work in the area software-based energy estimation in WSNs.

4.1 Motivation

While commonly used networking metrics such as packet delivery rate, source-to-sink latencies or maximum throughput can easily be determined in real-world WSN testbeds, measuring the power consumption of sensor nodes is much harder: costly high-resolution digital multimeters or cathode-ray oscilloscopes need to be hooked to the nodes in order to sample the varying low currents and voltages. For years, experimental research in the field of energy-aware and energy-conserving protocols

in distributed systems has required weeks or even months of tedious and time-consuming use of bulky cathode-ray oscilloscopes or high-resolution multimeters.

Customized devices such as the Sensor Node Management Devices (SNMDs) [78] have recently been developed as a cost-effective alternative to using high-frequency multimeters or oscilloscopes for *side-effect free* high-resolution energy measurement of sensor nodes (cf. Section 2.3.2 of Chapter 2). As SNMDs continuously stream the obtained current and voltage values over their USB port, a totally wired setup with reasonably powerful intermediate nodes (e.g., small desktop PCs or mesh nodes) to read and process this data is necessary. Such a setup, however, is usually not practicable and not economical in the case of large-scale experimental testbeds, and clearly unfeasible in case of outdoor WSN deployments.

4.2 Energy Estimation vs. Energy Measurement

Software-based energy estimation has been proposed by *Dunkels et al.* in [50] as a viable alternative to using costly hardware-based energy measurement equipment. The mechanism applied in [50] consists in bookkeeping the time the radio resides in the different transceiver modes and the time the CPU and the onboard sensors are used, on the node itself. Multiplying these times with previously determined power levels then leads to rough estimates for the consumed energy. This mechanism, along with similar studies on software-based energy-estimation, are discussed in more detail in Section 2.3.3 of Chapter 2. Many prominent protocol studies on Energy-Efficient MAC (E^2-MAC) protocols have entirely relied their experimental research results upon the same or a similar software-based approach for estimating the energy consumption of their protocol prototypes, e.g., the studies on S-MAC [190] or B-MAC [143]. More and more recent research papers utilize the same methodology, e.g., the studies of *Finne et al.* [65] or *Boano et al.* [21]. Yet, as already pointed out in [50], no existing study has yet validated the accuracy of this approach with physical hardware-based energy measurements.

Software-based energy estimation techniques clearly have their advantages and drawbacks. A purely software-based approach can only deliver estimates, which usually deviate more from the real energy consumption than hardware-based measurements. Software-based mechanisms further introduce inherent *side-effects*, as the estimation mechanism itself causes computational costs, which are hard to account for. The advantages of software-based energy-estimation mechanisms, however, are manifold: with an energy-estimation being present on the node at run-time, many energy-conserving WSN algorithms can finally be applied in real-world deployments. Many energy-conserving distributed algorithms rely on an estimation of the used and the residual energy endowment in order to take optimal decisions, e.g., in the context of routing, clustering or transmission power selection algorithms. With the WSN field moving from simulation-based towards real-world testbed-based research, finding a simple and painless, but yet accurate methodology for quick and reliable energy-estimation can be a significant milestone.

4.3 The Accuracy of Software-based Energy-Estimation

In the following, we discuss our approach to examine the *estimation accuracy* of different software-based energy-estimation models and methodologies. We empirically evaluate several energy-estimation models and parameter calibration methodologies with prototype implementations of IEEE 802.11-like CSMA and the three E^2-MAC protocols S-MAC, T-MAC and WiseMAC on the MSB430 sensor nodes. The MSB430 platform is discussed in detail in Section 2.1.3 of Chapter 2, the MAC protocols in Section 2.4 of the same chapter. We ran a large number of experiments under different traffic load levels and with different node instances, in order to statistically describe the resulting estimation accuracies.

Section 4.3.1 describes the experiment setup utilized throughout the entire empirical study. In Section 4.3.2, we discuss a discovered deviation effect among different instances of sensor nodes of the same type, which heavily impacts on the estimation accuracy. Section 4.3.3 then discusses the examined energy estimation models and their refinements, the parameter calibration methodologies, and elaborates on the accuracy of the resulting software-based energy-estimations using the different wireless channel MAC protocols.

4.3.1 Experiment Setup

We kept the measurement setup as simple as possible, in order to be able to repeatedly perform a significant number of experiment runs with different wireless channel protocols and traffic rates on the same experiment setup. We lay out nodes A, B, C with a distance of 30cm on a table, as depicted in Figure 4.1. Node *B* is hooked to an SNMD, which continuously measures its current and voltage. The SNMD devices are discussed in detail in Section 2.3.2 of Chapter 2. In order to determine the estimation accuracy, we needed to be able to simultaneously obtain both the software-based *estimations* and the unaffected physical hardware-based *measurements* of the same node *B* in each experiment. For this purpose, we had to keep node *B* unplugged from any serial interface, as the node would otherwise draw some small current from the powered USB serial interface cable. Hence, in order to obtain the software-based estimations of node *B* without accessing the node over a serial cable, we let node *B* write its energy estimation model data (time in transmit mode, time in receive mode, etc.) into the packet payload.

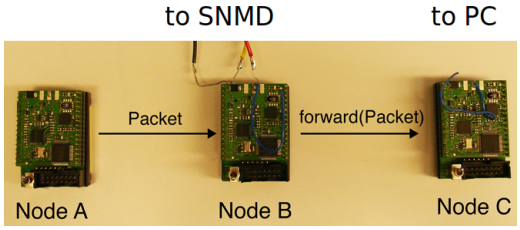


Fig. 4.1: Node A generating packets, Node B hooked to SNMD

Packets have a size of 50 bytes each (10 bytes header, 40 bytes payload). In each experiment run, node *A* starts sending Constant Bit Rate (CBR) traffic of rate *r* towards node *B* during $T_{exp} = 600s$. Right after the reception of the first packet, Node *B* starts keeping track of the time its transceiver resides in the different states. After injecting its estimation model data into the packet, node *B* forwards the packets to node *C*, which decapsulates the packet and logs node *B*'s energy estimation data to the serial interface, which is connected to a Desktop PC. During the entire experiment, the current trace of node *B* is read from the SNMD's serial interface, which is connected to the same Desktop PC. As discussed later in the analysis, we varied the traffic rate *r* at node *A* from very low rates (1 packet every 100s) to high rates (max. 2 packets/s) with each different wireless MAC protocol. We measured 10 independent runs for each setting, and evaluated 8 different node instances.

4.3.2 Hardware-dependent Deviations

Applying software-based energy estimation inevitably introduces inaccuracies. The differences between the *estimated* power consumption and the *physically measured* power consumption can generally be explained by two reasons: it might either stem from *hardware effects*, e.g., the slightly differing nodes' electronic hardware components characteristics, or from from the inherent imperfection of the *software-based model* and the applied estimation methodology. This section elaborates on the effect of the slight deviations of the power consumption of different node instances of the same node type. As discovered in previous experimental studies by *Landsiedel et al.* [106] and *Haratcherev et al.* [76], the power consumption of different instances often varies in the range of some few percent. [106] presumes that this variation stems from deviations in the electronic components tolerances. We hence first examined multiple instances of MSB430 nodes running different MAC protocols, given a constant traffic rate of 1 packet each 20s over $T_{exp} = 600s$. With this evaluation, we quantify the estimation inaccuracies caused by the variation in the energy consumption of different instances of the same node type - in our case the MSB430 platform.

Figure 4.2 depicts the energy consumed by eight different instances of MSB430 nodes and the four examined protocols during 10 experiment runs. Each bar depicts

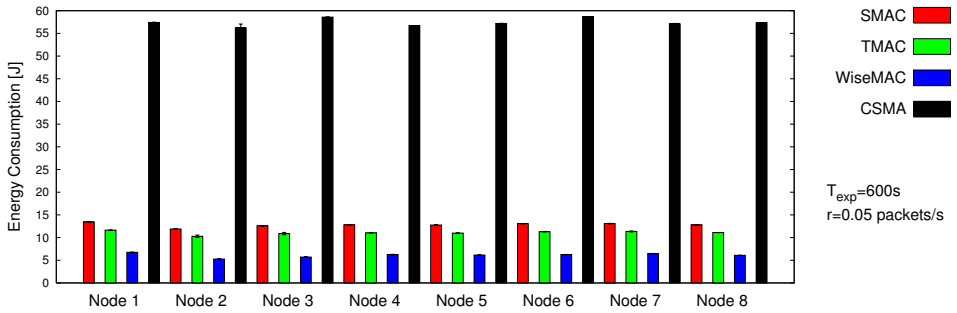


Fig. 4.2: Energy consumed by Eight different Instances of MSB430 Nodes

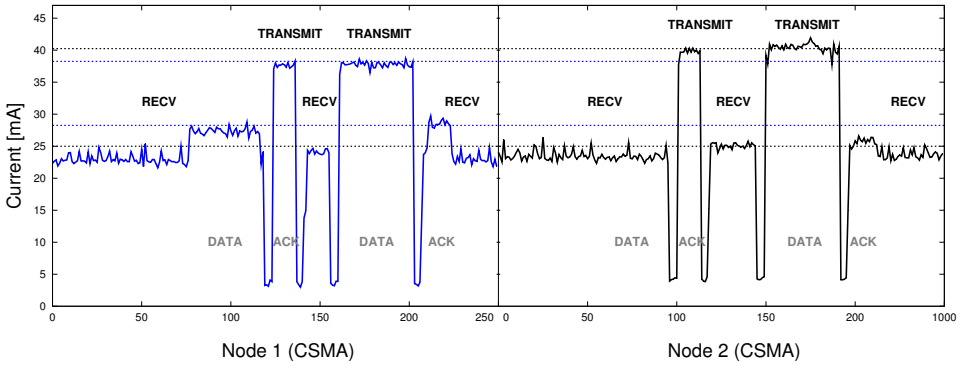


Fig. 4.3: Current Draw of Nodes 1 and 2 running CSMA

the mean value and standard deviation measured during 10 independent runs - the latter was low in most cases and is hence barely visible. The energy consumption obviously varies heavily from protocol to protocol (eg. WiseMAC vs. CSMA). The variation from node to node is also clearly visible, e.g., the energy consumed by node 6 running CSMA is roughly 4% higher than that of node 2.

We investigated the reason for these differences in the current traces and found that indeed, the current drawn from different nodes can vary to a certain degree, and that the variation can even differ for each of the different transceiver states. Figure 4.3 depicts the current traces of nodes 1 and 2 running CSMA and receiving a data packet, and sending it further to another node. As one can clearly see, node 1 draws approximately 2 mA less than node 2 when transmitting. Although the transmission power settings were set identically for all nodes, the current levels in the transmit state obviously varied to a certain degree. A further anomaly we encountered is that some nodes drew more current in receive mode when actually receiving data compared to listening to an idle channel, whereas in most cases, no significant difference between these two cases could be measured. This effect is visible in Figure 4.3 as well: node 1 consumes approximately 3 mA more when receiving data, compared to node 2, which consumes more or less the same current when receiving data or listening to an idle channel. As both nodes are running the same interrupt service routine code and did not run any other computationally intensive tasks during this time, the CPU can neither be held accountable for this effect. We further discovered slight differences in the peak energy consumption as well as in the duration of transceiver switches depending on the protocol, and even depending on the traffic load.

We presume that these differences stem from the inaccuracies in the production of the electronic components. Fast switching between the different operation modes of the radio could probably also have a temporary impact on the behavior of active circuit elements. Although the temperature is known to impact on the power consumption of electronic devices, we can safely exclude this as an explanation for the discovered deviations, as all experiments were run under room temperature in the same laboratory environment.

Statistical Characterization of Node Deviations

In an attempt to quantify the discovered differences between the eight measured node instances, we a) determined the mean and standard deviation of the measured energy consumptions of all measurement runs of all eight nodes for the CSMA protocol in the experiment described in 4.3.1 (with $T_{exp} = 600s$ and a traffic rate *r* of 0.05 packets/s), and b) compared each pair of nodes to determine the maximally differing nodes. We chose CSMA because at examined traffic rates, no packet loss occurred within all CSMA runs. Hence, the CSMA experiment runs were most suited for examining the per-node differences.

The *mean consumed energy* of the eight different nodes throughout T_{exp} was 57.55 Joules with a standard deviation of 1.54%. Hence, roughly two thirds of all node instances exhibit a value in between 57.55 Joules $\pm$ 1.54%, given that the variation between different nodes follows normal distribution. We conjectured that the latter is the case, as Jarque-Bera's test on the normality of the measurement variation (JB-value: 0.701) could not be rejected, cf. *Draper and Smith* [43]. The Jarque-Bera test, cf. equation (Jarque-Bera), is a statistical test of whether empirical data has the skewness and kurtosis matching a normal distribution. JB is defined as:

$$JB = \frac{n}{6}\left(S^2 + \frac{1}{4}(K-3)^2\right) \qquad \text{(Jarque-Bera)}$$

where the variable n denotes the number of samples. The variables S and K denote the samples' skewness and kurtosis, the third and fourth central momentums of the samples' distribution.

$$S = \frac{\hat{\mu}_3}{\hat{\sigma}^3} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{3/2}} \qquad \text{(Skewness)}$$

$$K = \frac{\hat{\mu}_4}{\hat{\sigma}^4} - 3 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^2} - 3 \qquad \text{(Kurtosis)}$$

The *maximum deviation* between the mean measured energy consumption of the two maximally differing nodes was determined to be 4.24% (of the respective higher value). We tested the claim that these two nodes do actually differ significantly from each other, i.e. that the discovered deviations are not caused by coincidence or the limited set of observations. We found that the null-hypothesis of a two-sided t-test claiming that the two nodes exhibit the same mean energy consumption (=on average consume the same amount of energy) could safely be rejected at the 95% confidence level. However, this was not the case for all the node pairs, as some groups of nodes obviously exhibit similar patterns in their energy consumption, as clearly visible in Figure 4.2, e.g., in the CSMA bars. *Benjamin Nyffenegger* provided a more detailed analysis of the pairwise deviations in his thesis [136].

4.3.3 Software-based Energy-Estimation Models

In this section we analyze the impact of the choice of the estimation model on the resulting estimation accuracy, using experiments with different traffic load levels and different wireless channel protocols. With the variation between different nodes being in the range of more than 4% for the specified experiment scenario, we decided to use exactly *the same sensor node* and also the *same SNMD device* throughout the entire analysis in this section, in order not to introduce variations caused by differing measurement hardware and measured sensor node hardware.

Three States Model (TSM)

The most frequently used model to date for estimating a node's energy consumption consists in modeling the latter as a function of the three states of the radio transceiver *receive/idle listening*, *transmit* and *sleep*, cf. [76][190][143]. We henceforth refer to this model as the *Three States Model*. The Contiki OS' energy estimation mechanism models the radio's power consumption using this model, but *separately* tries to keep track of the CPU power consumption, which can vary depending on the Low-Power-Mode (LPM) it is currently operating. The ScatterWeb^2 OS used in this study puts the CPU to LPM1 as soon all events have been processed, where the node's current is approximately 1.8 mA, given that the radio is turned off. With the CPU active and the radio off, the node current is roughly 3.5 mA. As our examined E^2-MAC protocols generally do not incur intensive computations, we neglected to account for the CPU costs separately, and considered the CPU's power consumption to be *integrated* within the three states of the transceiver. Estimating the CPU power consumption in software when applying E^2-MAC protocols is anyway not easy to achieve, as most of the MAC-related CPU activity takes place in interrupt service routines. Accounting for such

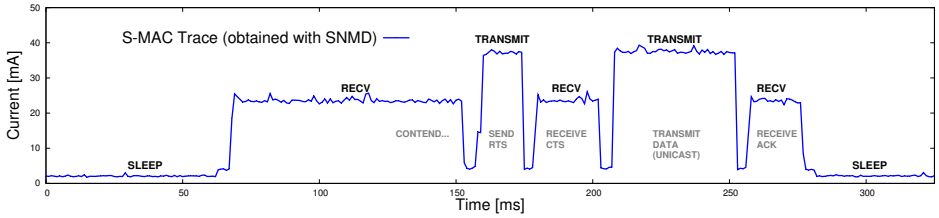


Fig. 4.4: Current Draw of node B

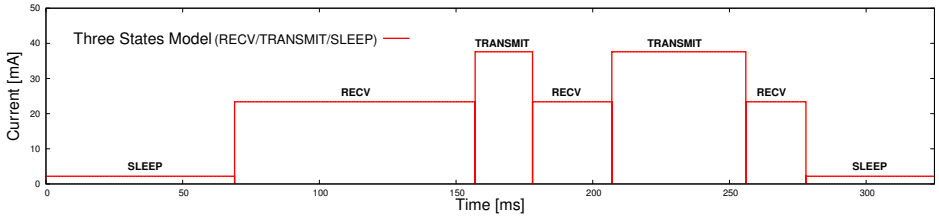


Fig. 4.5: Current modeled by the *Three States Model (TSM)*

may even cause more costs than the protocol-related computations themselves, cf. *Landsiedel et al.* [106]. If the CPU activity does not vary much across state changes of the radio transceiver, modeling the CPU and radio integrally safely holds. Figure 4.4 illustrates that for the given E^2-MAC protocol, accounting for CPU in a combined manner with the three different power levels of the radio is sufficient.

We henceforth modeled the energy consumption of our S-MAC, T-MAC, WiseMAC and CSMA implementations using the abovementioned *Three States Model*. We let the nodes keep track of the time differences between the transceiver switches, in order to determine how much time has been spent in each state. Figure 4.4 depicts the current draw during the active interval of an S-MAC frame containing an RTS/CTS handshake and a subsequent data packet transmission. Figure 4.5 illustrates how this current draw is being approximated by the *Three States Model*. The total energy consumed (denoted as E) corresponds to the area below the current draw multiplied by the supply voltage, which is assumed to be constant in the model. Analytically, the *Three States Model* can be formulated as equation (TSM). The consumed energy E is calculated as the power level of the node in the receive state P_{rcv} multiplied by the total time spent in this state T_{rcv}, and the respective terms for the transmit and sleep states ($P_{slp}T_{slp}$ and $P_{tx}T_{tx}$). This approach is identical to the one applied in [76], [190] and [143].

$$E = P_{rcv}T_{rcv} + P_{tx}T_{tx} + P_{slp}T_{slp} = I_{rcv}V_{rcv}T_{rcv} + I_{tx}V_{tx}T_{tx} + I_{slp}V_{slp}T_{slp} \quad \text{(TSM)}$$

The studies [190], [143], [25] and [76] calibrate the parameters of their energy model by measuring the currents the nodes draw in the different states, and multiplying it with the supply voltage to obtain P_{rcv}, P_{tx} and P_{slp}. They do so by using either oscilloscopes or high-precision multimeters and by measuring the current in each state over a certain timespan. In the first attempt, we pursued exactly the same approach, and determined the mean values of I_{rcv}, I_{tx}, I_{slp} by measuring each

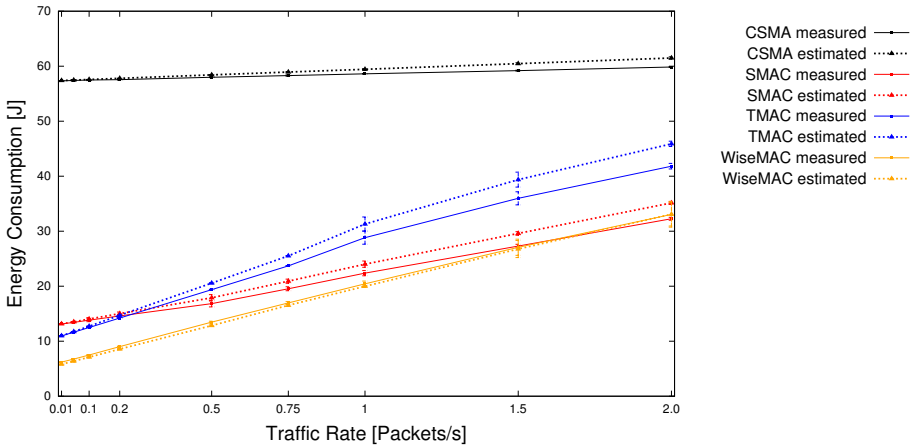


Fig. 4.6: Measured vs. Estimated Energy Consumption

state of the *measurement* node using the SNMD for a couple of seconds. The stable mean values were determined to be 23.5353 mA, 37.4872 mA and 2.1495 mA for I_{rcv}, I_{tx}, I_{slp}, respectively. We further set the voltage according to the supply voltage of the SNMD to $V_{rcv} = V_{tx} = V_{slp} = 4.064V$.

Figure 4.6 depicts the mean values of the energy measurements and the estimations being computed with the *Three States Model* - using the parameters for P_{rcv}, P_{tx} and P_{slp} measured in the example trace. One can clearly see that the estimations fit quite well for low traffic rates, but that the gaps between mean estimations and mean measurements become larger with higher rates of packets being sent over the *measurement* node. For most protocols - especially S-MAC and T-MAC - the energy estimation over-estimates the energy consumed by the node with increasing load. This increasing over-estimation stems from the fact that the *Three States Model* does not account for the transceiver switches. As one can clearly see comparing Figure 4.4 with Figure 4.5, the current draw decreases to roughly 4 mA when the transceiver is switched to receive or transmit - hence drawing less current than estimated with the *Three States Model*. By defining parameters through example measurement, the impact of the applied traffic load and the frequent transceiver switches as well as the particularities of the MAC protocol are not being taken into account at all. Extrapolating from a short example measurement of a node hence leads to suboptimal parameters for the *Three States Model*, even when using the same node for parameter calibration and the evaluation of the accuracy.

Parameter Definition through Ordinary Least Squares (OLS):

Being able to physically measure the current draw of a sensor node *and* at the same time obtain the software-based estimation calculated by the node itself offers the opportunity to relate the estimations to the real-world measurements. Using the plethora of experimental data gained in the many experiments runs (in total over 12 GB), we reflected upon a method to determine more resilient parameters for the unknown variables P_{rcv}, P_{tx}, P_{slp} of the *Three States Model*. Ideally, the software-based energy estimation running on the node should neither rely on the particularities of a specific MAC protocol, nor on the shape or intensity of the traffic. *Ordinary Least Squares (OLS) Regression Analysis* yielded the most suitable technique to determine the unknown variables for a linear estimation model with multiple unknown variables. OLS finds the model parameters that minimize the sum of squared errors (SSE) between the estimations and observations (=the real-world energy measurements captured with the SNMD devices).

We formulated a multivariate OLS regression model with the *explanatory variables* T_{rcv}, T_{tx}, T_{slp} (the times spent in the different transceiver states, calculated at runtime), as well as the physically measured *dependent variable* E obtained using the SNMD device. The resulting estimation equation (OLS-I) hence simply comprises equation (TSM) and the error term ε for the residuals.

$$E = P_{rcv}T_{rcv} + P_{tx}T_{tx} + P_{slp}T_{slp} + \varepsilon \qquad \text{(OLS-I)}$$

We define the matrix $\boldsymbol{X}$ containing all the observations of the *explanatory variables*, consisting in the three columns T_{rcv}, T_{tx}, T_{slp} with a row for each measurement. We further define the vector $\boldsymbol{y}$ containing the corresponding observations of the *dependent variable*, i.e., the energy measured using the SNMD. The unknown parameters P_{rcv}, P_{slp}, P_{tx} that are to be determined are defined as the three-dimensional vector $\boldsymbol{\beta} = (P_{rcv}, P_{slp}, P_{tx})$.

$$\boldsymbol{X} = \begin{bmatrix} T_{\mathrm{slp}_1} & T_{\mathrm{rcv}_1} & T_{\mathrm{tx}_1} \\ T_{\mathrm{slp}_2} & T_{\mathrm{rcv}_2} & T_{\mathrm{tx}_2} \\ \vdots & \vdots & \vdots \\ T_{\mathrm{slp}_n} & T_{\mathrm{rcv}_n} & T_{\mathrm{tx}_n} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \qquad \boldsymbol{\beta} = \begin{bmatrix} P_{rcv} \\ P_{slp} \\ P_{tx} \end{bmatrix}$$

With the variables above, the model can be specified in vector-matrix form as:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{MVOLS}$$

The deviation of the estimation of the energy-consumption $\hat{\boldsymbol{y}}$ from the real-world measured values (with the SNMD) $\boldsymbol{y}$ forms the vector of the residuals:

$$\boldsymbol{\varepsilon} = \hat{\boldsymbol{\varepsilon}} = \boldsymbol{y} - \hat{\boldsymbol{y}}$$

We intend to find the parameters $\hat{\boldsymbol{\beta}}$, for which the estimated energy consumption values $\hat{\boldsymbol{y}}$ minimize the sum of squared errors from the measured values $\boldsymbol{y}$. Transforming (MVOLS) yields the following:

$$\boldsymbol{\varepsilon} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$$

The sum of squared errors (SSE) can conveniently be written in vector form as:

$$\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon} = \sum_{i=1}^{n} \varepsilon_i^2$$

Hence, the vector $\boldsymbol{\beta}$ we seek has to satisfy the condition

$$\begin{aligned} \min_{\boldsymbol{\beta}}(\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon}) &= (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})'(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \\ &= (\boldsymbol{y}' - \boldsymbol{X}'\boldsymbol{\beta}')(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \\ &= \boldsymbol{y}'\boldsymbol{y} - \boldsymbol{\beta}'\boldsymbol{X}'\boldsymbol{y} - \boldsymbol{y}'\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\beta}'\boldsymbol{X}'\boldsymbol{X}\boldsymbol{\beta} \end{aligned}$$

Setting the differential of $\frac{d(\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon})}{d\beta}$ to zero is the condition the minimum has to satisfy:

$$\begin{aligned} \frac{d(\boldsymbol{\varepsilon}'\boldsymbol{\varepsilon})}{d\beta} &= -2\boldsymbol{X}'\boldsymbol{y}' + 2\boldsymbol{X}'\boldsymbol{X} - 2\boldsymbol{X}'\boldsymbol{\beta} = 0 \\ &\Rightarrow \boldsymbol{X}'\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}'\boldsymbol{y} \end{aligned}$$

The OLS estimator $\hat{\boldsymbol{\beta}}$ containing the parameters $(\hat{P_{rcv}}, \hat{P_{tx}}, \hat{P_{slp}})$ that minimize the sum of squared errors hence calculates as:

$$\hat{\boldsymbol{\beta}} = ((\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}')\boldsymbol{y}$$

We assessed the coefficient of determination R^2 to measure the *goodness of fit* of the multivariate linear regression model and obtained a surprisingly high value of $R^2 = 0.9980$. The coefficient of determination measures the proportion of variability in a data set, which can be explained by the regression model. It is defined as:

$$R^2 = 1 - \frac{SSR}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \qquad \text{(CoeffDet)}$$

Estimation Accuracy of the Three States Model:

In order to determine the accuracy of the OLS-calibrated model, a cross-validation with totally new experimental data is inevitable to omit overfitting effects, cf. *Draper and Smith* [43]. The determination of the parameters P_{rcv}, P_{tx}, P_{slp} using OLS regression was hence achieved on a first set of experiment runs, the so-called *training set*. The estimation accuracy results of this section were then gained with a new set of experimental data, to which we will further refer as *validation set*. We fed $\hat{\boldsymbol{\beta}}$ containing the OLS estimators of the unknown variables $\hat{P_{rcv}}, \hat{P_{tx}}, \hat{P_{slp}}$ into the node's estimation model and estimated the energy consumption with the validation set. We considered the so-called *Mean Absolute Error (MAE)*, the average difference between the estimations and the measured values, cf. equation (MAE) to be the best statistical measure for the *accuracy* of the employed *Three States Model*. The MAE and its standard deviations calculated across all protocols and traffic rates in the validation set (henceforth always given as percentage of the SNMD-measured values) is depicted in Figure 4.7. For each traffic rate, the estimation error using the OLS estimator parameters is 4.2% to 35.9% lower than the corresponding error when using the model parameters defined through example measurement. Across all measurements, the mean absolute estimation error and standard deviation (denoted as $\mu \pm \sigma$) of the *Three States Model* with the parame-

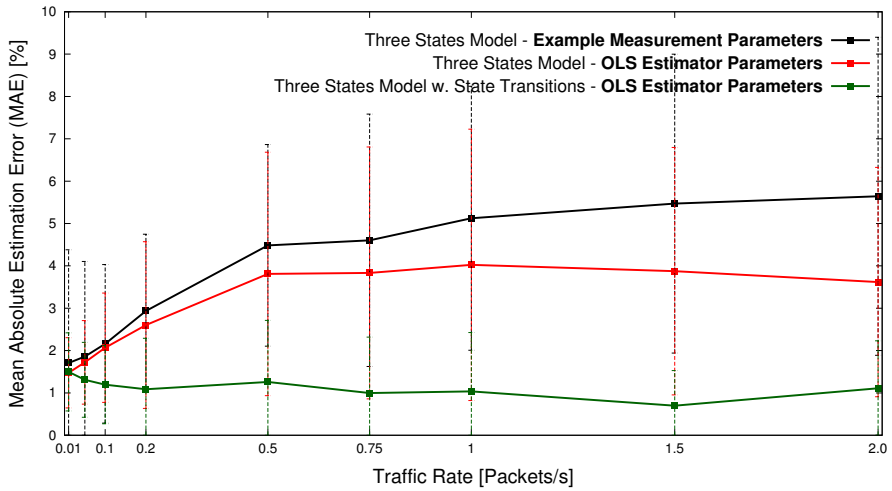


Fig. 4.7: Mean Absolute Estimation Error (in %) vs. Traffic Rate (packets/s)

ters defined by example measurement equals $3.77\% \pm 3.17\%$. When determining the parameters by OLS, we obtain $3.00\ \% \pm 2.55\%$ - hence achieving an overall reduction of the MAE by 21% only by altering the calibration technique. Taking into account the entirety of experimental data by application of OLS to the calibration set hence clearly pays off in the resulting estimation accuracy.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|\varepsilon_i| \qquad \text{(MAE)}$$

Three States Model with State Transitions (TSMwST)

With the mean absolute estimation error still in the range of 3% or more, we investigated further means to improve the estimation accuracy. As Figure 4.8 exhibits, the current draw temporarily drops to approximately 4 mA during the state switches. These state switches remain unaccounted for in the OLS regression model specified in equation (OLS-I). We first attempted to sum up the transition times between the transceiver states. However, this approach led to unsatisfactory results, as the ScatterWeb^2 OS only supports a clock in milliseconds precision. Yet, the approach of simply counting the transceiver switches and integrating them into the OLS regression model led to a significant improvement in the estimation accuracy. The number of transceiver switches (*from* an arbitrary state) *to* the *receive*, *transmit* or *sleep* state was accounted for with the additional regressands s_{rcv}, s_{tx}, and s_{slp}. We refer to this model as *Three States Model with State Transitions* hereafter, as specified in equation (TSMwST). Figure 4.9 illustrates the model's concept of a node's current draw.

$$E = P_{rcv}T_{rcv} + P_{tx}T_{tx} + P_{slp}T_{slp} + \alpha s_{rcv} + \beta s_{tx} + \gamma s_{slp} \qquad \text{(TSMwST)}$$

According to this enhanced model, the energy consumed by an arbitrary node is a function of the total time it has its radio transceiver in the three different states (de-

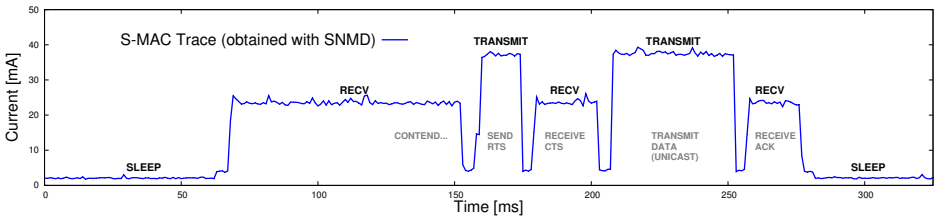


Fig. 4.8: Current Draw of node B

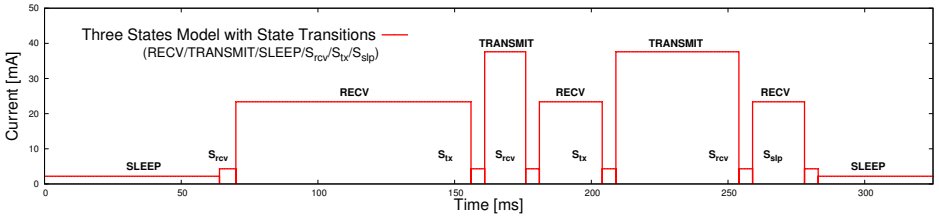


Fig. 4.9: Current modeled by the *Three States Model with State Transitions (TSMwST)*

noted as T_{rcv}, T_{tx}, T_{slp}) and the three adjustment terms αs_{rcv}, βs_{tx}, and γs_{slp}. The parameters α, β, γ compensate for the transceiver switches to the states *receive*, *transmit* and *sleep*.

Parameter Definition through Ordinary Least Squares (OLS):

We derived the OLS regression equation (OLS-II) to the model (TSMwST) with the *explanatory variables* T_{rcv}, T_{tx}, $T_{slp}, s_{rcv}, s_{tx}, s_{slp}$, as well as the *dependent variable* E (of which we obtain the *measured value* with the SNMD) as

$$E = P_{rcv}T_{rcv} + P_{tx}T_{tx} + P_{slp}T_{slp} + \alpha s_{rcv} + \beta s_{tx} + \gamma s_{slp} + \varepsilon \qquad \text{(OLS-II)}$$

The OLS estimator $\hat{\boldsymbol{\beta}} = (\hat{P_{rcv}}, \hat{P_{tx}}, \hat{P_{slp}}, \hat{\alpha}, \hat{\beta}, \hat{\gamma})$ is calculated in analogy to Section 4.3.3. For this purpose, the matrix $\boldsymbol{X}$ in the OLS model in matrix form (cf. equation (MVOLS)) has to be redefined, the vectors $\boldsymbol{y}$ and $\boldsymbol{\beta}$ remain unchanged:

$$\boldsymbol{X} = \begin{bmatrix} T_{\mathrm{slp}_1} & T_{\mathrm{rcv}_1} & T_{\mathrm{tx}_1} & s_{rcv_1} & s_{tx_1} & s_{slp_1} \\ T_{\mathrm{slp}_2} & T_{\mathrm{rcv}_2} & T_{\mathrm{tx}_2} & s_{rcv_2} & s_{tx_2} & s_{slp_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ T_{\mathrm{slp}_n} & T_{\mathrm{rcv}_n} & T_{\mathrm{tx}_n} & s_{rcv_n} & s_{tx_n} & s_{slp_n} \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} P_{rcv} \\ P_{slp} \\ P_{tx} \end{bmatrix}$$

We obtained a coefficient of determination of $R^2 = 0.9998$ for the multivariate linear regression model (OLS-II), a slightly higher value than for OLS-I. However, when comparing the *goodness of fit* of two regression models, the R^2 indicator is not a meaningful criterion, as it never decreases when adding more regressands. The *adjusted coefficient of determination* $\bar{R}^2$ adjusts for the number of explanatory terms in a model. Unlike R^2, this coefficient only increases when the increase of explanatory variables actually improves the model. An increase of $\bar{R}^2$ upon addition of an explanatory variable to a multivariate OLS model is hence generally understood as a proof that the new model delivers a better fit to the measured data. Equation (CoeffDetAdj) defines the adjusted coefficient of determination $\bar{R}^2$. The variable n denotes the number of the observations, the variable k the number of explanatory variables, which increased from 3 to 6 in our case.

$$\bar{R}^2 = 1 - \frac{SSR}{TSS}\frac{(n-1)}{(n-k)} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}\frac{(n-1)}{(n-k)} \qquad \text{(CoeffDetAdj)}$$

An even better coefficient for comparing the *goodness of fit* of two regression models is the Akaike Information Criterion (AIC), as specified in (AIC). The variable k denotes the number of parameters of the model, and θ the maximized likelihood function for the estimated model, cf. *Draper and Smith* [43]. The lower the AIC value, the better the fit of the empirical data to the specified model. We measured the $\bar{R}^2$ and AIC coefficients before and after adding the transceiver switches s_{rcv}, s_{tx}, s_{slp}, to the OLS model (OLS-I vs. OLS-II).

$$AIC = -2\ln(\theta) + k \qquad \text{(AIC)}$$

With $\bar{R}^2$ increasing from $\bar{R}^2_I = 0.9801$ to $\bar{R}^2_{II} = 0.9980$, and AIC decreasing from $AIC_I = 2.5036$ to $AIC_{II} = 0.2154$, we can safely claim that the *Three States Model with State Transitions* delivers a significantly better fit to the measurement data than the today's most widely used simple *Three States Model*.

Estimation Accuracy of the Three States Model with State Transitions:

We calibrated the OLS estimators for the parameters of the second model with the *training set*, and examined the resulting estimation accuracy on the *validation set*. Across all measurements, the MAE and standard deviation (denoted as $\mu \pm \sigma$) of the software-based estimations using the *Three States Model with State Transitions* (and the parameters determined by OLS) compared to the physically measured values equals $1.13\% \pm 1.15\%$. Comparing this result to the $3.00\ \% \pm 2.55\%$ obtained with the *Three States Model* (and the parameters determined by OLS), our proposed model enhancement led to an overall reduction of the MAE by remarkable 62.3%, as also illustrated in Figure 4.7.

The Impact of Calibration on the Estimation Accuracy

This section evaluates the impact of different possible granularities of *calibration* on the achievable accuracy of the software-based energy estimation technique. Throughout this section we henceforth utilize the same multivariate OLS regression methodology and the *Three States Model with State Transitions* as described in Section 4.3.3, as applying this model generally led to the lowest estimation errors.

Per-Node Calibration: Different wireless sensor node instances often exhibit a slightly different behavior with respect to their power consumption levels in the different transceiver states. This effect has been observed in previous studies [106] [76], and has been quantified for the utilized MSB430 platform in Section 4.3.2. We have encountered node pairs of the same node type that differed by more than 4% in their physically measured energy consumption. Hence, even the best *node-generic* software-based energy estimation mechanism can be more than 4%, if its underlying model parameters were not calibrated on a *per-node* basis.

Researchers intending to calibrate their energy estimation model with only one particular sensor node instance must therefore be aware that their energy consumption estimates will deviate from the *real* energy consumption by the unavoidable hardware-based variation, unless each node has previously been calibrated individually. However, calibrating on a *per-node* basis means that *every single node* needs to be physically measured (e.g., with an SNMD or a high-resolution multimeter) ideally with different MAC protocols and different traffic rates. Only this time-intensive calibration leads to the set of *per-node* but *protocol-generic* estimation model parameters which has been shown in Section 4.3.3 to reduce the mean absolute estimation error ($\mu \pm \sigma$) to $1.13\% \pm 1.15\%$.

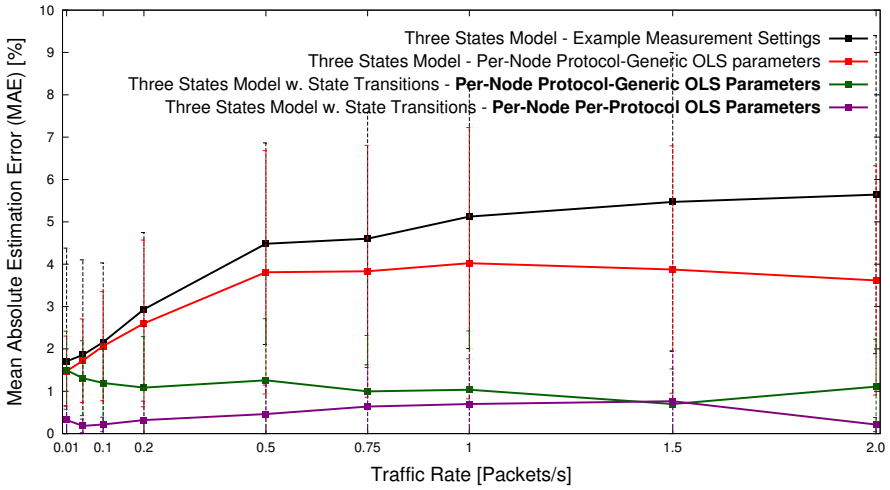


Fig. 4.10: Mean Absolute Estimation Error (in %) vs. Traffic Rate (packets/s)

Per-Node and Per-Protocol Calibration: In Section 4.3.3, we intentionally *generalized* from the particularities of the MAC protocol by running OLS over four different MAC protocols. Hence, we obtained protocol-independent (but node-specific) estimation parameters. In order to obtain *per-protocol* (and node-specific) calibrated estimator parameter values, the methodology applied in Section 4.3.3 can be applied without much adaptation. Basically, the methodology has to be applied four times independently for each protocol. In each case, only the observations of the specific protocol and node have to be chosen from the training set in order to calculate the OLS estimator $\boldsymbol{\beta} = (P_{rcv}, P_{slp}, P_{tx}, s_{rcv}, s_{slp}, s_{tx})$, which results in four different estimators $\boldsymbol{\beta}_{WiseMAC}$, $\boldsymbol{\beta}_{S-MAC}$, $\boldsymbol{\beta}_{T-MAC}$ and $\boldsymbol{\beta}_{CSMA}$. For each different protocol, the estimation model then needs to be aware of the applied MAC protocol at run-time, and must be equipped with the numerical values of all four parameter vectors $\boldsymbol{\beta}$. The same *specialization* effect can also be achieved by supplying more information to the OLS model by introducing *dummy variables* that indicate the currently used protocol, cf. *Draper and Smith* [43] p.299ff.

We propose *per-protocol* calibration as an even more accurate estimation approach, which might be useful if researchers know exactly what protocol they intend to use on the MAC layer in advance. We calculated different OLS parameter sets for each of the four protocols (S-MAC, T-MAC, WiseMAC, CSMA) and used the same node (node 1 in Figure 4.2) used in Section 4.3.3 for calculating the resulting accuracy on the validation set. The combined approach of *per-node and per-protocol* calibration obviously leads to the highest accuracy. Across all four protocols and traffic rates, we obtained a mean estimation error and standard deviation ($\mu \pm \sigma$) of only $0.42\% \pm 0.72\%$. The combined calibration approach, however, has multiplicative impact on the overhead before network deployment, as all nodes need to be equipped with tailor-made estimation model parameters for each protocol. Figure 4.10 illustrates the different estimation errors when applying the *per-node and protocol-generic* or the *per-node and per-protocol* calibration approach.

4.4 Conclusions

In this chapter, we have described our efforts towards a reliable and robust methodology for *software-based energy estimation*. In this study, we identified and quantified the different factors, which cause deviations of the software-based estimations from the *real* physically measurable energy consumption.

In Section 4.3.2, we have observed and quantified the impact of *hardware-related* deviations in the energy consumption patterns, which impact on the accuracy of software-based energy estimations when universal and generic parameters are used instead of per-node calibrated values. In Section 4.3.3, we have thoroughly examined the accuracy of different software-based models and calibration techniques. We have calculated the estimation errors in large set of experiments run across a wide range of parameters, i.e., nine different traffic volumes, four different wireless channel MAC protocols and eight different instances of the MSB430 platform.

Our main findings can be summarized as follows: We have conveyed that software-based energy estimation can be a valuable alternative to using sophisticated measurement hardware, especially in outdoor- deployments where the latter is impossible - at least for evaluating protocols where the CPU is used frugally, i.e., E^2-MAC or routing protocols. The inaccuracies in the production of the electronic components have been shown to impact on different power consumption levels, which led to nodes differing by more than 4% in their energy consumption over an experiment duration of only 10 minutes. Enhancing today's most widely used simple *Three States Model* with information regarding the state transitions and applying multivariate OLS regression to calibrate the model parameters has been shown to remarkably reduce the estimation error. The mean absolute error (MAE) and standard deviation ($\mu \pm \sigma$) of the energy estimations by the software-based model using protocol-generic but per-node calibrated parameters could be pushed to as few as $1.13\% \pm 1.15\%$. Applying even more sophisticated parameter calibration of per-node *and* per-protocol calibration has been shown to reduce the mean absolute error and standard deviation to as few as only $0.42\% \pm 0.72\%$ across the four evaluated wireless channel MAC protocols S-MAC, T-MAC, WiseMAC, and IEEE 802.11-like CSMA. For both calibration methodologies, the protocol-generic/per-node-calibrated and the per-protocol/per-node-calibrated approach, the resulting MAE did neither depend on the employed traffic rate nor the MAC protocol, and did not exceed 1.6% and 0.75%, respectively. The proposed methodologies are hence *robust* against the choice of the MAC protocol and the traffic rate.

The library functions, as well as the developed estimation models and calibration techniques for estimating the energy of a sensor node at run-time have been further used for the evaluations of Part II of this thesis. Relying on the work presented in this chapter and the TARWIS testbed management system (cf. Chapter 3), we could reliably quantify the energy consumption of not only a single node, but the entire testbed network running our energy-efficient and traffic-adaptive MAC protocol prototypes in Chapter 7.

Part II

Contributions To Communication Protocols

Chapter 5

A Traffic-Adaptive Extension for WiseMAC

In this chapter we evaluate the performance of the WiseMAC [57] burst transfer mode and propose an enhanced scheme, which addresses the problem of bottleneck nodes in tree-based scenarios. The study published in [87][91][89] ignited and strengthened our interest on throughput-maximizing but at the same time energy-efficient mechanisms on the MAC layer, which finally led to the more sophisticated traffic adaptivity mechanisms designed, implemented and evaluated in Chapter 7.

We thoroughly analyze the WiseMAC mechanism to cope with packet bursts called *More Bit*, as well as our proposed extension, both in simulation and on a real-world sensor node platform. Section 5.1 motivates our investigations by outlining the problem we address. Section 5.2 describes the basic WiseMAC protocol with the existing *More Bit* mechanism. Section 5.3 then introduces our proposed *Extended More Bit* mechanism. Section 5.4 evaluates both approaches in simulation (Section 5.4.1) and by means of small-scale real-world experiments with a prototype (Section 5.4.2). Section 5.5 concludes the chapter.

5.1 Motivation

WiseMAC [57] is to date one of the most established asynchronous preamble-sampling energy-efficient MAC (E^2-MAC) protocols, and has had a big influence on its successors (e.g. X-MAC [25], ContikiMAC [48]). The sound performance and high efficiency of WiseMAC under various traffic conditions has been independently pointed out in *Langendoen et al.* [107] and our own contributions [85][90]. Among the preamble-sampling-based protocols, it must be seen as the most efficient approach, since it only employs a minimal preamble for collision avoidance and clock drift compensation, as opposed to B-MAC [143] or X-MAC [25], which send out long preambles before every payload frame transmission. The recent survey of nine E^2-MAC protocols [107] points out that "the WiseMAC protocol showed a remarkable consistent behavior across a wide range of operational conditions, always achieving the best or second-best performance". With sparse traffic,

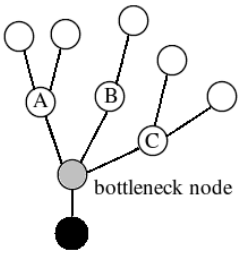


Fig. 5.1: Bottleneck

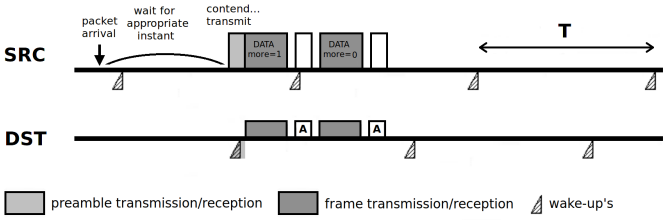


Fig. 5.2: WiseMAC More Bit

WiseMAC comes close to the theoretically achievable lower bounds of energy-efficiency in case of unicast point-to-point transmissions, cf. *El-Hoiydi* [56]. However, the high energy-efficiency of WiseMAC comes at the cost of a limited maximum throughput and packet loss occurring with rather low traffic rates already, e.g. due to congestive situations where multiple nodes intend to transmit to the same gateway node. If many sensors simultaneously detect and report data to the base station, gateway nodes receiving traffic from several sources become throughput-restraining bottleneck nodes, as depicted in Figure 5.1 with nodes A,B,C forwarding packets from large subtrees to the grey shaded bottleneck node.

5.2 WiseMAC More Bit Scheme

WiseMAC suggests an optional fragmentation scheme called *More Bit* in [56], which succeeds in increasing the maximum achievable throughput across one link. The scheme consists in setting a flag (the *More Bit*) in a unicast MAC frame whenever the sender has more packets to send to the same destination. The *More Bit* in the frame header signals to the receiving node not to turn off the transceiver after receiving the frame, but to switch to receive again after transmitting the frame acknowledgement in order to receive the next packet in line (cf. Figure 5.2). When a sender node has multiple packets to send to the same destination, it hence does not need to wait for the next wake-up of the receiver for each frame, but can transmit all packets in one burst, which increases the achievable WiseMAC throughput.

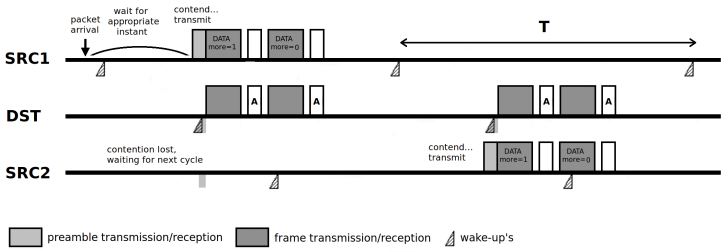
The scheme proved to be effective in scenarios with varying traffic, especially with packet bursts generated by single nodes. However, the more bit scheme only includes one sender and one destination. Basically, it only attempts to improve the traffic adaptivity along one link. The improvement in traffic adaptivity is therefore rather limited to point-to-point scenarios. In large multi-hop WSN topologies, however, the typical situation is that nodes closer to the base station need to forward data from large sub-trees. As schematically depicted in Figure 5.1, such bottleneck nodes have to forward messages received by many other child nodes and their subtrees, e.g., nodes A, B and C. The *More Bit* scheme does not help at all if several nodes aim to simultaneously transmit packets to the same bottleneck node. If each node has one or a few packets pending for the bottleneck node, one node after the other will have to wait for one particular wake-up of the bottleneck node. The *More*

Bit scheme achieves that one node after the other can transmit a burst of packets, but the duty-cycle of the receiving bottleneck node is left untouched.

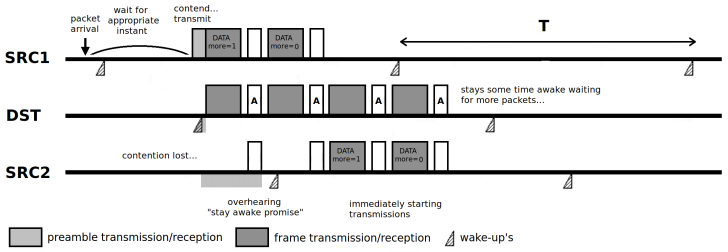
5.3 Extended More Bit Scheme

In order to increase the effectiveness of the *More Bit* in scenarios of multiple sources transmitting to one bottleneck node, we developed the *Extended More Bit* scheme. We extended the semantics of the *More Bit* to become a *promise* for *staying awake* for a full base interval T. When a node hence receives a frame with the *More Bit* set, it *promises* to all nodes waiting to forward traffic to remain awake by setting the *Extended More Bit* in the acknowledgement.

Figure 5.3 depicts both schemes, the WiseMAC *More Bit* 5.3 (a) scheme as well as our proposed *Extended More Bit* 5.3 (b). The figure compares how the mechanisms react in a situation where two sources SRC1 and SRC2 simultaneously need to transmit some packets to the same node DST, possibly because an event has occurred in their vicinity. If SRC1 and SRC2 both attempt to reach DST in the same wake interval, the contention mechanism of WiseMAC will decide who is first. SRC1 wins the contention and sends its first two frames with the *More Bit* set. With the *Extended More Bit* displayed in Figure 5.3, the destination node acknowledges the *More Bit* in the acknowledgement and stays awake for at least one base wake interval T. As SRC2 has lost the contention, it will wait and overhear the transmission from SRC1 to DST. By hearing the *Extended More Bit* in the acknowledgement, SRC2 knows that it can start sending its own data frames right



(a) WiseMAC More Bit Mechanism



(b) Extended More Bit mechanism

Fig. 5.3: WiseMAC More Bit and Extended More Bit

after SRC1 has finished its transmissions. The advantage of the *Extended More Bit* scheme is that no time is wasted with DST switching to sleep while other nodes are still buffering packets destined to it. Notice that the transmission of SRC2 can start immediately after the transmissions of node SRC1. If a node requests its destination to stay awake for one next packet (as done in the *More Bit* scheme), the receiver node treats this request as an indication of increased load. The scheme is not applied after every unicast transmission, but is activated whenever bursts of packets occur, as transmissions of single packets are frequent in WSNs and do not yet indicate increasing load.

5.4 Experimental Evaluation

5.4.1 Simulation-based Evaluation

We simulated the two illustrated mechanisms *More Bit* and *Extended More Bit* in a scenario of 90 nodes uniformly distributed across an area of 300m x 300m in the OMNeT++ Network Simulator [178]. We used the default wireless channel model (Free Space) of the Mobility Framework [44]. As done in most simulation studies on the energy consumption of MAC protocol-related mechanisms, we calculated the energy consumption of the sensor nodes with a state-based energy model according to time spent in three operation modes *sleep*, *receive* and *transmit*, weighted with the respective energetic costs taken from the transceiver specs, as usually done in network-simulator based evaluations of WSN MAC protocols, e.g., the frequently cited studies [38][72][107].

We applied the transceiver parameters of the TR1001 radio transceiver [156] (transmission rate, state transition delays, power consumption), which is the radio of our prototype node platform. The parameters of the simulation environment, energy consumption model and transceiver specific settings, as well as WiseMAC-specific parameters are listed in Table 5.1. Traffic is generated during 1 hour at each node

Path Loss Coefficient α	3.5	Sensitivity	-101.2 dBm
Carrier Frequency	868 MHz	Carrier Sense Sensitivity	-112 dBm
Transmit Power	0.1 mW	Communication Range	50 m
SNR Threshold	4 dB	Carrier Sensing Range	100 m
Transceiver Parameters [156]:			
Supply Voltage	3 V	Recv to Transmit	12 μs
Transmit Current	12 mA	Transmit to Recv	12 μs
Recv Current	4.5 mA	Sleep to Recv	518 μs
Sleep Current	5 μA	Recv to Sleep	10 μs
WiseMAC Parameters:			
Base Interval T	250 ms	Maximum Retries	3
Duty-Cycle	5%	Packet Size	20 bytes
Packet Queue Length	15		

Table 5.1: OMNeT++ Simulation Parameters

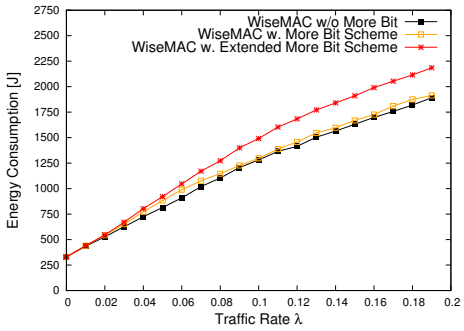


Fig. 5.4: Energy Consumption

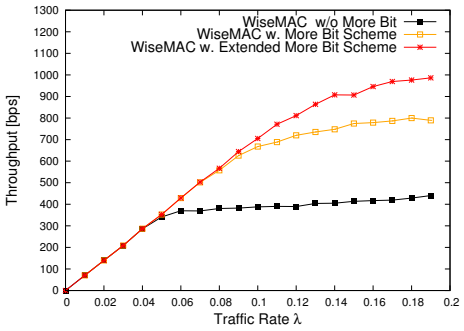


Fig. 5.5: Throughput

(except for the sink) according to a Poisson process with increasing traffic intensity λ, and sent towards one single sink in the lower left corner. Nodes apply static shortest path routing, and are assumed to know their hop-count optimal gateway. They forward their packets over the latter during the entire simulation run, picking one at random in case more than one convey the same hop distance to the sink.

Figure 5.4 depicts the aggregated energy consumption of the entire network at the end of the simulation, and Figure 5.5 the payload throughput received at the sink station. As one can clearly see in Figure 5.4, WiseMAC (without the *More Bit*) is both energy-efficient and somewhat traffic-adaptive for rates of λ in between $[0, 0.05]$. With no traffic, the energy consumption remains very low. With linear increase of traffic, WiseMAC reacts with a more or less linear increase of the energy consumption. Throughput stalls at a traffic rate of $\lambda = 0.05$ already. Without the *More Bit*, nodes only transmit one packet per wake-up, the throughput hence depends on the duration of the basic wake interval T.

As one can clearly see in Figure 5.5, the introduction of the WiseMAC *More Bit* option improves the throughput by a factor of roughly 2. The ability to transmit a burst of buffered packets to the intended receiver allows for reaching a much higher throughput. The *More Bit* somewhat improves the traffic awareness of the WiseMAC protocol, as it allows for achieving a higher maximum throughput with a still reasonably high efficiency for rates of λ in between $[0, 0.1]$, but remains efficient for low traffic. Figure 5.5 further conveys that an increase of the maximum throughput is possible with the *Extended More Bit*. As nodes stay awake for a certain time interval after receiving a packet burst, even if no other node needs to transmit packets, the improved throughput comes with slightly increased energy costs. Considering the ratio of throughput and energy, the *Extended More Bit*, however, performs even better than the *More Bit* scheme for high traffic ($\lambda \geq 0.1$).

5.4.2 Evaluation on Embedded Sensor Boards

In order to examine and compare the real-world feasibility of the simulated MAC mechanisms and WiseMAC extensions, we implemented the WiseMAC protocol a) without *More Bit*, b) with *More Bit* and c) our proposed *Extended More Bit* scheme on Embedded Sensor Boards (ESB) [158] on top of the ScatterWeb (v1.0) OS [159].

Base Interval T	500 ms	Medium Reservation Preamble	uniform [0,6] ms
Duty-Cycle	1%	MAC Header	13 bytes
Baud Rate	19'200	Payload	12 bytes
Bit Rate	9'600 bps	Packet Queue Length	20
Minimum Preamble	5 ms	Maximum Retries	3

Table 5.2: WiseMAC on ESB nodes: Prototype Parameters

The ESB is equipped with the MSP430 microchip [171], various sensors and the 868.35 MHz transceiver TR1001 [156]. The platform is discussed in more detail in Chapter 2. Sensors and radio transceiver can be turned on and off, which results in different levels of energy consumption. The parameters listed on Table 5.2 led to robust functioning of WiseMAC on the ESBs.

Preamble Sampling and Frame Transmission: The ESB WiseMAC prototype implements the main features of the protocol as described in [57]. In each wake-up, the node senses the channel. If it does not recognize a preamble byte within its duty-cycle of $\triangle t = T \times 1\% = 5\ ms$, it turns its radio off until the next wake-up. If it recognizes the preamble byte, it keeps the radio on until preamble and frame are correctly received. When a packet has to be sent, the so-called *network handler* determines whether the receiver is already *known* and its schedule offset is already stored in the WiseMAC schedule offset table. If this is the case, the node waits for the receiver's wake-up and then transmits preamble and frame. In case the receiver is yet unknown, transmission immediately starts with a preamble of duration T.

Evaluation of Throughput

We measured the throughput of WiseMAC without *More Bit*, with *More Bit* and our proposed *Extended More Bit* when generating constant traffic of equal rate from two senders (SRC1, SRC2) towards one receiver (DST), basically the same scenario as depicted in Figure 5.3. When both senders concurrently forward packets to the receiver, the receiver with its limited wake-ups becomes a bottleneck. The *Extended More Bit* alleviates the impact of this problem.

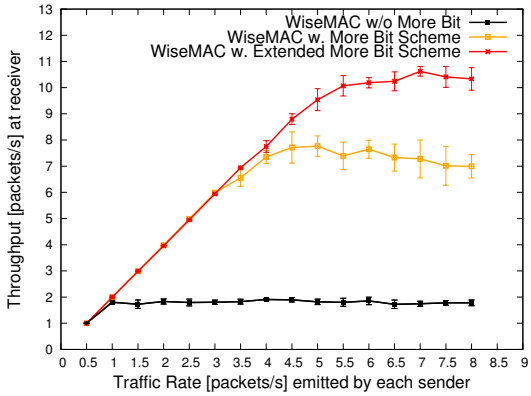


Fig. 5.6: Throughput for WiseMAC, More Bit and Extended More Bit on ESBs

Figure 5.6 depicts the mean throughput measured in 10 experiment runs for each setting. Obviously, the WiseMAC protocol without the *More Bit* can only deliver one packet per wake-up, and therefore, the packet transmission rate is limited to two packets per second ($=\frac{1}{T}$). When increasing the rate, packets are subsequently queued in the buffer and dropped when the buffer is full. When two stations apply the *More Bit* scheme, they can alternately empty their transmit buffers with a burst of packets. The sending station receives packets from its application layer and buffers them until the receiver node's next wake-up. The sender then transmits frames with the *More Bit* set, listens for the acknowledgement and continues sending the next packet in line, until its buffer is empty. Applying the *Extended More Bit* scheme increased the throughput to much higher values. Figure 5.6 clearly conveys that the *Extended More Bit* scheme is superior to the original WiseMAC *More Bit* with respect to the achieved throughput by more than 25%. The superior performance of roughly 20% has been found similar in both simulation and real-world experiments, although the topology setup was much simpler. Yet, the effect of the bottleneck nodes became clearly measurable in both scenarios, and delivered astonishingly similar results.

5.5 Conclusions

In this chapter we evaluated the WiseMAC burst transfer mode *More Bit* and our proposed *Extended More Bit* in simulation and on a real-world sensor platform. The results confirm that the *Extended More Bit* basing on the receiver *promising* to remain awake after a packet burst performs better than the original WiseMAC *More Bit* scheme with respect to the achievable throughput. The superior performance of 20%-25% has been found similar in both simulation and real-world experiments.

The evaluation of the results presented in this chapter basically form a starting point in our investigations on throughput-maximizing but at the same time energy-efficient mechanisms on the MAC layer, which are discussed in depth in Chapter 7. In terms of time, this chapter by far describes the oldest results among the contributions presented in this thesis. When experimenting with the radio duty-cycling MAC protocol mechanisms on ESB nodes in the context of this rather preliminary study, we realized that indeed, the average energy consumption could be drastically reduced when applying preamble sampling and a duty-cycle of only 1%. However, the price to be paid was that of severely degraded Quality of Service: while sending shell commands to sensor nodes was practicable with ScatterWeb's simple CSMA protocol, manual interaction with the WiseMAC prototype across several hops became a cumbersome and tiring issue. We investigated whether there were yet mechanisms proposed that would allow nodes to temporally abandon their wake-up pattern when services with higher Quality of Service constraints (e.g, real-time interaction) had to be handled. However, we found that no existing protocol targeted at such scenarios and that in general, research had focused solely on energy-efficiency, leaving any Quality of Service aspects aside.

In the subsequent chapter, we examine a selection of the most frequently cited E^2-MAC protocols with respect to their ability to handle traffic with variable load. With this evaluation, we intend to profoundly clarify *which* mechanisms, and *which* class of protocols seemed to be the best starting point to accomplish the traffic-aware behavior on the MAC level we have just discovered to be missing.

Chapter 6

Energy Efficient MAC Protocols under variable Traffic Conditions

In this chapter, we evaluate today's most well-known Energy-Efficient Medium Access Control (E^2-MAC) protocols under variable traffic patterns, in order to obtain a first impression how the *state of the art* in MAC protocols behaves in these situations. In almost any study on E^2-MAC protocols published in the recent past, traffic has been assumed to be *low* and at *constant rate*, e.g., the studies on S-MAC [190], B-MAC [143], T-MAC [38] or WiseMAC [57] all evaluate their protocols under low and Constant Bit Rate (CBR) traffic. The study on X-MAC [25] varies the number of transmitters, but not the traffic rate over time.

As pointed out in Section 5.5 of Chapter 5, the issue of *adaptivity* and *flexibility* of MAC protocols with respect to timely variable traffic has yet only been superficially studied. There has been no clear notion yet how to *assess* or *measure* traffic adaptivity in the literature. The quote of Lord Kelvin, the inventor of the absolute thermodynamic temperature scale saying that "if you can't measure it, you can't improve it" is still valid in almost any problem domain. In order to improve traffic adaptivity in E^2-MAC protocols, we first require a consistent notion of this property. A comparative analysis of the state of the art in E^2-MAC protocols with respect to their adaptability towards variable traffic is definitely yet missing. With the evaluation presented in this chapter and published in [90], we close this missing gap and thoroughly evaluate the energy-throughput and energy-latency trade-offs of a selection of today's most prominent E^2-MAC protocols. Based on our findings, we motivate the need for traffic-adaptive MAC protocol mechanisms.

In Section 6.2, we experimentally examine the most well-known E^2-MAC protocols for WSNs proposed between 2002 and 2008 in scenarios with variable traffic conditions. Sections 6.2.1-6.2.2 first outline the preliminaries of our simulation analysis, i.e., the examined protocols and their parameters, the simulator environment, the wireless channel model and its parameters, the experiment setup and the shape of the varying traffic under which the protocol models have been evaluated. Section 6.2.3 analyzes how the throughput and the energy-consumption of today's most well-known protocols react to variable load. Section 6.2.4 then examines

the energy-throughput and energy-latency trade-offs of each protocol. Section 6.3 introduces an intuitive definition for the ability of a protocol to adapt to varying traffic load at run-time - defining our understanding of *traffic adaptivity* with an unambiguous notion of a formal three-partite metric. By applying this metric to experimental results of Section 6.2.4, we can finally compare the state of the art in today's E^2-MAC protocols with respect to their run-time *traffic adaptivity* and motivate our future contributions in this area. Section 6.4 concludes this chapter.

6.1 Motivation

Energy-efficiency is a major concern in the design of WSN communication protocols. As the radio transceiver typically accounts for a major portion of a WSN node's power consumption, E^2-MAC protocols attempt to maximize the time the radio is kept turned off, in order to minimize the overall energy expenditures. However, with duty-cycling the radio, these protocols trade off a higher energy-efficiency versus classical Quality of Service parameters (e.g., throughput, latency, reliability). Today's E^2-MAC protocols are able to deliver little amounts of data with a low energy footprint, but introduce severe restrictions with respect to throughput and latency. Often, throughput is reduced to a fraction of that of energy-unconstrained MAC protocols.

With WSNs growing in popularity in industry, applications and use cases have emerged where higher throughput and reasonable Quality of Service are required during periods of intense network activity - e.g., in the case of Wireless Multimedia Sensor Networks (WMSNs) transmitting an image when an event is detected, cf. *Akyildiz et al.* [4] and *Almalkawi et al.* [7]. Further use cases can be found in health-care related sensor networks registering anomalies in patient's sensor readings that need to be transmitted to a central node, cf. the studies on *CodeBlue* [119] and *Chipara et al.* [35]. As pointed out in [7], an E^2-MAC protocol for such applications should "maximize network throughput", but "be energy-efficient" for most of the time. It should therefore reduce the major sources of energy-waste, but still offer reasonable Quality of Service (high throughput, low delay) in case of increasing network activity. Similar concepts of adaptive resource allocation have been successfully implemented in many other domains of distributed computing, often with the goal of conserving energy, e.g., dynamic frequency/voltage scaling implemented in the CPU chips of most of today's mobile devices.

6.2 Simulation-based Evaluation

In order to examine the behavior of the current state of the art in E^2-MAC protocols under variable traffic load, we went on to filter out the most frequently cited protocols in the E^2-MAC field. We chose to implement the S-MAC [190], T-MAC [38], B-MAC [143], WiseMAC [57] and X-MAC [25] protocol, along with the reference protocols IdealMAC and simple energy-unconstrained IEEE 802.11-like CSMA in

the OMNeT++ Network Simulator [178]. More detailed information about the protocols can be found in Section 2.4 of Chapter 2. We used the OMNeT++ extension Mobility Framework (MF) [44], which supports simulations of wireless ad hoc and mobile networks on top of OMNeT++.

6.2.1 Simulation Model

In the following, we briefly discuss the particularities of the simulation setup and the examined protocol models with the most crucial parameters settings.

IdealMAC

The concept of an ideal MAC protocol for WSNs, to which we refer to as *IdealMAC* hereafter, plays a key role in our approach to measure and quantify the E^2-MAC protocol run-time traffic adaptivity. This concept of an idealized MAC layer has been used by *El-Hoiyidi* [57][56] to show where the *lower bounds* of E^2-MAC protocol efficiency are. IdealMAC models the physical constraints of E^2-MAC protocols, such as the channel bandwidth, the delays and costs of the transceiver switches. The key point of the concept is that IdealMAC assumes that there is *no information asymmetry* between sender nodes and receiver nodes. Nodes always *know* when they need to switch to receive/transmit in order to handle transmissions. IdealMAC is depicted in Figure 6.1 where a source node A transmits a frame via an intermediate node B to a destination node C.

In IdealMAC, nodes are always asleep in case of no traffic. At the same instant a sender node receives a packet from the upper layer, such as node A in Figure 6.1, the receiver node B instantly switches its radio transceiver from *sleep* to *receive*, because it also *knows* that the source node wants to transmit. After frame reception, the intermediate node B forwards the frame to destination node C in the same manner. By assuming perfect global knowledge among all participating nodes, IdealMAC has the lowest-possible delay any E^2-MAC protocol, and the highest possible energy-efficiency. It exhibits no overhead for periodic duty-cycling, periodic synchronization. As nodes immediately turn their transceivers to sleep after frame transmissions, they do not suffer from overhearing or idle listening. It is further assumed that nodes can always avoid collisions without introducing an RTS-CTS exchange. Nodes always *know* whether their targeted receivers are ready to receive messages or whether their neighbors are currently occupying the channel.

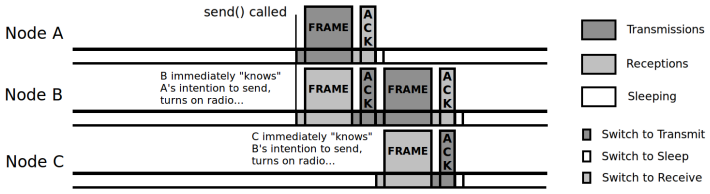


Fig. 6.1: IdealMAC Reference Protocol

Wireless Propagation Model

In the recent years, network simulation tools and simulation studies in general have been heavily criticized, mainly because of oversimplified simulation model assumptions and inadequate parameter settings [105][11]. Unit-Disk-Graph (also referred-to as *Flat-Earth*) based simulation models, as well as the deterministic Free Space Model have been shown to even produce misleading results, cf. *Kotz et al.* [103]. The Free-Space model, as specified in *Rappaport* [153], calculates the received power P_r at a distance d from the sender according to the transmission power P_t, the antenna gains of the transmitter and the receiver G_t,G_r, the wavelength λ and the so-called system-loss factor L.

$$Pr(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}$$

Even though the model parameters can be varied, the resulting received power is completely deterministic. Two nodes with a certain distance are either within each other's transmission range or not, given that there are no concurrent transmissions. In order to more realistically reflect the characteristics of wireless propagation (high packet error rate due to signal attenuation, multipath propagation, shadowing, fading and similia), we applied the Log-Normal Shadowing Model [153], which has been recently introduced into OMNeT++ [104]. This channel model allows for a more realistic simulation of wireless channel properties and phenomena. It models small-scale shadowing and fading effects for each frame transmission by adding a random perturbation factor to the reception power. The perturbation factor follows a log-normal distribution with a user-selectable deviation σ. The received power $Pr_{LogNormal}$ at a distance d is a random variable following the log-normal distribution:

$$Pr_{LogNormal}(d; \sigma^2) \sim ln(Pr(d), \sigma^2)$$

The specified SNR thresholds then determine whether a packet arriving with a received power of $Pr_{LogNormal}$ is correctly received or not. Figure 6.2 depicts

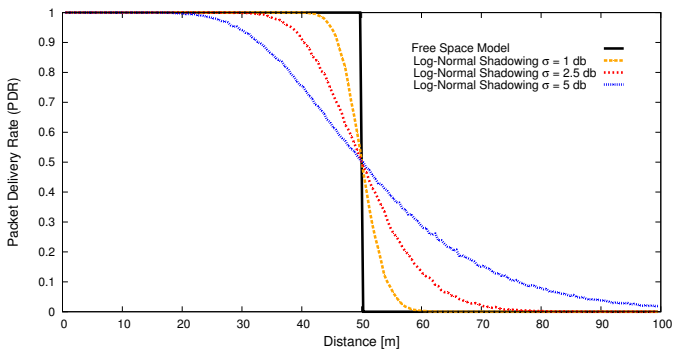


Fig. 6.2: Free Space vs. Log Normal Shadowing

the packet delivery rate (PDR) (y) versus the distance (x) between a sender and a receiver, applying the default OMNeT++ Free Space Model and comparing it to the Log-Normal Shadowing Model with different values of σ. With the Free Space Model, the received power is a simple deterministic function of the distance. When having only one sender at a time, the reception probability immediately drops from 100% to 0% when the distance between the sender and the receiver exceeds 50m (with the given transmission power and SNR threshold settings). Using the Log-Normal Model, the PDR decreases gradually with the distance, exhibiting different slopes with different values of the deviation σ of the random perturbation factor.

Still, any simulation environment for WSNs must be considered incomplete, since crucial wireless phenomena and hardware-related properties are only modeled artificially and many side-effects are left out. However, with choosing an adequate channel model and suitable environment settings, we intended to at least stick to the *best practices* of wireless network simulations.

Energy Model

We modeled the power consumption of the sensor nodes with a state transition model with respect to the time spent in three operation modes sleep, receive and transmit, weighted with the respective energy costs, as usually done in network-simulator based evaluations of WSN MAC protocols, e.g., [38][72][107]. We further used a simple linear battery model, which does not take self-discharge into account. Table 6.1 lists current, voltage and transmission rate of the CC1020 [173], a byte-level radio transceiver in the 804-940 MHz ISM frequency band. The CC1020 is used by the MSB430 sensor nodes platform [14], which we later use in Chapter 7 as the main platform for prototyping maximally traffic-adaptive E^2-MAC protocols on real-world sensor nodes. Table 6.1 further lists the parameters of the wireless channel model of the OMNeT++ Network Simulator [178], as well as experiment-specific settings (e.g., header size, payload, simulation time).

Transceiver CC1020 [173]:			
Transmit Current I_{tx}	21.9 mA	Supply Voltage U	3 V
Recv Current I_{rx}	17.6 mA	Transmission Rate R	115'200 bps
Sleep Current I_{sleep}	1 μA		
OMNeT++ Parameters:			
Path Loss Coefficient α	3.5	Transmit Power	0.1 mW
Log-normal Deviation σ	2.5 db	SNR Threshold	4 dB
Carrier Frequency	868 MHz	Sensitivity	-100.67 dBm
Carrier Sense Sensitivity	-112 dBm		
Simulation Settings:			
Simulation Runs	100	Simulated Time	1000 s
Maximum ARQ Retries	3	Frame Header Size	14 bytes
Payload	50 bytes		

Table 6.1: Simulation and Experiment Parameters

CSMA	Contention Window CW	10 ms
S-MAC	Listen Interval Duty-Cycle	[100, 200, 300, 500, 1000, 2000] ms 10%, 20%
T-MAC	Frame Length Contention Window CW SYNC size D_{SYNC} RTS size D_{RTS} CTS size D_{CTS} Timeout SYNC period	[50, 100, 200, 300, 500, 1000] ms 5 ms 14 bytes 14 bytes 10 bytes $1.5 \times (CW + D_{RTS}/R + D_{CTS}/R)$ 10 s
B-MAC	Base Interval Duty-Cycle Medium Reservation Interval	[25, 50, 100, 200, 500] ms [8, 4, 2, 1, 0.4] % uniform (0,10) $\times\ t_{rx-tx}$
WiseMAC	Base Interval Duty-Cycle Medium Reservation Interval	[25, 50, 100, 200, 500, 1000] ms [8, 4, 2, 1, 0.4, 0.2] % uniform (0,10) $\times\ t_{rx-tx}$
X-MAC	Max Interval Min Interval Early-ACK size D_{EACK} Inter-Strobe-Interval Listen Interval	[100, 200, 500] ms 10 ms 10 bytes D_{EACK}/R $D_{EACK}/R + t_{rx-tx} + t_{tx-rx}$

Table 6.2: E^2-MAC Protocol Parameters

Protocol Simulation Models

Table 6.2 displays the main simulation parameters of the simulated E^2-MAC protocol models. As the protocol behavior often heavily depends on the choice of the essential protocol parameters (e.g., basic wake interval, duty-cycle, etc.), we studied the protocols with different *configurations* of those parameters, by varying the parameters over a wide range. Instead of only comparing single parameter settings of two protocols, we choose multiple parameter sets for each protocol, which gives a deeper insight into each protocol algorithm's advantages and disadvantages.

For the *slotted* protocols S-MAC and T-MAC, we assume that the nodes' wake-up intervals are synchronized at the beginning of the experiment (as assumed in many MAC studies, e.g., in the study [57] examining WiseMAC and comparing against T-MAC). In X-MAC [25], we integrated a simple traffic adaptation algorithm that sets the duration of the wake/sleep intervals as the inverse of the incoming packet rate, but remains in between the parameters [Max Interval, Min Interval] specified in Table 6.2. This adaptation algorithm specified in [25] has been implemented in the initial X-MAC prototype on top of the MANTIS OS [20], but not in the widely used Contiki X-MAC layer [170]. WiseMAC [57] implements a cheap collision avoidance and hidden node prevention mechanism that relies on each node having an *extended carrier sensing range* ($\sim$ 2$\times$*transmission range*), i.e., each node is assumed to be able to perceive transmissions of nodes are outside of its transmission range by observing an increase in the noise level of the channel. Such a

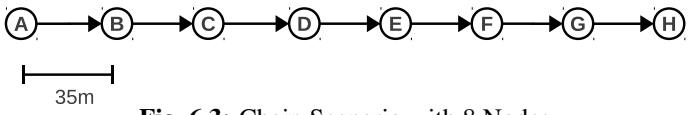


Fig. 6.3: Chain Scenario with 8 Nodes

mechanism is assumed to be implementable on most of today's radio transceivers, e.g., by observing the Received Signal Strength Indication (RSSI) value and setting appropriate thresholds. RSSI is an indication of the power level being received by the antenna. Most of today's radio transceivers provide an RSSI value in order to give an indication of the received signal strength to the link layer.

In order to allow for a fair comparison, we implemented a packet burst transfer mode for each simulated E^2-MAC protocol, such that nodes can transmit queued packet trains in a burst. Nodes can signal that they have pending packets to the receiver and continue transmitting packets in a burst, receiving an acknowledgment for each frame.

6.2.2 Experiment Setup

We simulated a chain consisting of 8 nodes with the source node A generating load, which is then forwarded hop-by-hop towards the sink node H, similarly as done in the studies on S-MAC [190] and B-MAC [143]. Almost any study on E^2-MAC protocols applies Constant Bit Rate (CBR) traffic during one simulation run. In contrast to this, we varied the offered traffic from low rates to high rates and back during each simulation run, as the major interest in this chapter consists in quantifying the adaptivity of the examined protocols at run-time.

Figure 6.4 displays the offered load generated at the application layer of the source node A. Traffic is low (0.1 packets/s) for most of the time, but there are load peaks during which the packet rate is linearly increased up to 22 packets/s and then again linearly decreased to the low level. We chose 22 packets/s as the load maximum as this had proved to be the maximum throughput that CSMA could handle without major packet loss across the 8 nodes chain with the parameters specified in Tables 6.2 and 6.1. When increasing the rate above the rate of 22 packets/s, throughput stalls and all additional packets are either dropped due to buffer overflows or are lost due to collisions with competing transmissions. As depicted in

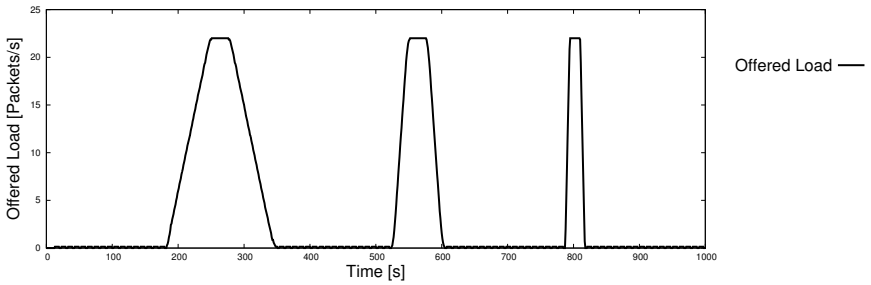


Fig. 6.4: Offered Load (Packets/s)

Figure 6.4, the load is increased linearly, with three different slopes for the load increase. In the first peak, the load is increased from 0.1 packets/s to 22 packets/s within 60s, and decreased over the same time period, hence giving the protocols some time to continuously adapt to the change in traffic volume. In the second peak, load is increased and decreased faster (20s), and in the third peak almost instantaneously (5s). Using these different slopes for the variation in the offered load, we study how the existing E^2-MAC protocols react to slowly and/or rapidly varying traffic conditions. Numerically, the slopes for the three load peaks amount for $\pm$ 0.36, $\pm$ 1.1 and $\pm$ 4.4 packets/s^2 for the load increase and decrease periods, respectively.

6.2.3 Network Throughput and Network Power Consumption

Figures 6.5 and 6.6 displays the rate of received packets at the sink node H vs. simulation time. All subsequent curves are averaged from 100 simulation runs for each protocol. As one can clearly see comparing the received packets in Figures 6.5 and 6.6 with the offered load in Figure 6.4, IdealMAC manages to handle all packets from source to the sink. CSMA only suffers minor packet loss at the load peaks. The S-MAC protocol with its static fixed-duration listen interval only manages to handle up to roughly 3 packets/s. T-MAC, B-MAC, WiseMAC and X-MAC reach higher throughput rates at the load peaks, but their throughput stalls at maximum 8 packets/s, which corresponds to $35 - 40\%$ of that of CSMA.

Figures 6.5 and 6.6 clearly convey that the current state of the art in E^2-MAC pro-

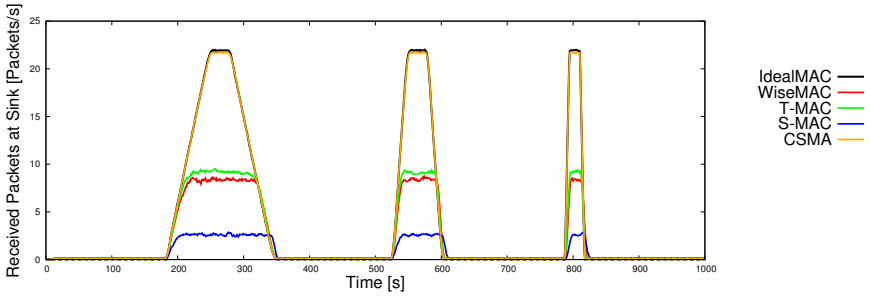


Fig. 6.5: Received Packets - S-MAC, T-MAC, WiseMAC vs. IdealMAC, CSMA

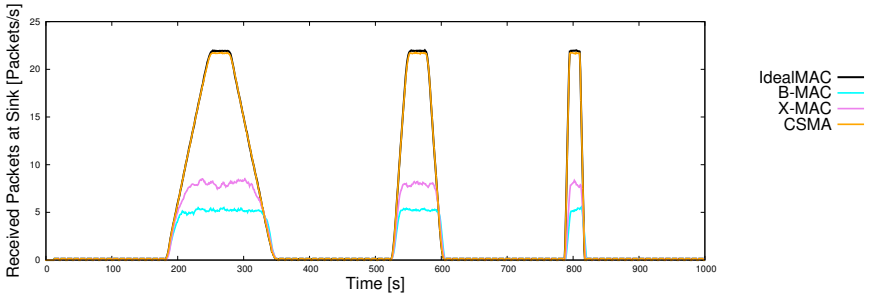


Fig. 6.6: Received Packets - B-MAC, X-MAC vs. IdealMAC and CSMA

tocols is not yet sufficiently adaptive with respect to varying load conditions, as the protocols generally do not manage to adapt their behavior to the load conditions. Although there is sufficient channel capacity, they do not manage to respond to the increased load with allocation of the respective resources needed to handle this imposed load. The radio transceiver is still turned off too aggressively and/or used inefficiently by the E^2-MAC protocols in case of load peaks. In contrast to this, many other adaptive resource allocation mechanisms in computing systems succeed in alternating between the minimum and the maximum performance that the underlying hardware offers. For example, in dynamic voltage/frequency scaling, adaptive mechanisms stepwise alternate between a low voltage/frequency mode and the *maximum* voltage/frequency the CPU supports.

Figures 6.7 and 6.8 depict the aggregated power consumption of all eight sensor nodes' radio interfaces versus simulation time. One can clearly see the big gap between the E^2-MAC protocols and energy-unconstrained CSMA. With low traffic, CSMA wastes a lot of energy on idle listening. The load peaks are hardly visible at all, as the transceiver does not use much more power when sending and receiving data, compared to idle listening. In contrast to CSMA, the IdealMAC reference protocol illustrates the ideal behavior of an E^2-MAC protocol, allocating as much energy as needed to handle the imposed load, and immediately deallocating it with decreasing load. The figures convey that the examined E^2-MAC protocols already exhibit an adaptive behavior with respect to the power consumption. They all consume less power with low traffic rates, and more at the load peaks. However, they clearly differ in the level of power consumption and thus their energy-efficiency in the sparse-traffic case (cf. S-MAC vs. WiseMAC), but also in the

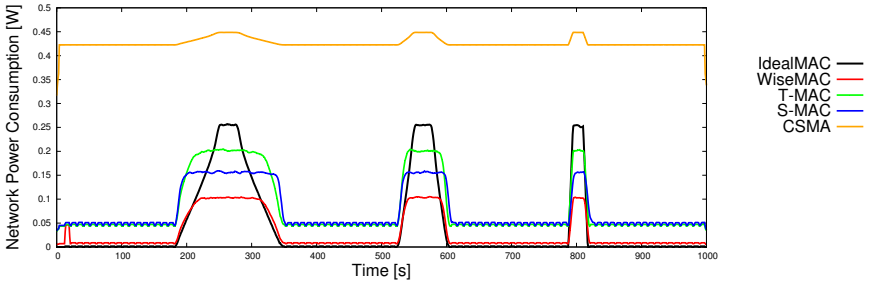


Fig. 6.7: Power Consumption - S-MAC, T-MAC, WiseMAC vs. IdealMAC, CSMA

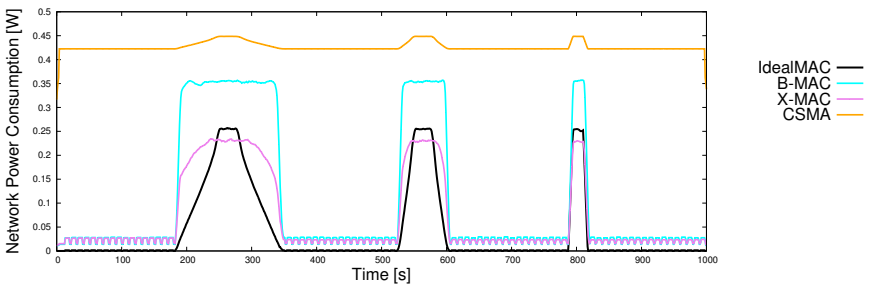


Fig. 6.8: Power Consumption - B-MAC, X-MAC vs. IdealMAC, CSMA

reaction to the linearly increasing and decreasing load level. The *slotted* E^2-MAC protocols S-MAC and T-MAC exhibit a much higher power consumption at r=0.1 packets/s, due to their overhead to keep their sleep-wake intervals synchronized. With WiseMAC or B-MAC, nodes do not maintain common sleep-wake schedules. Nodes only wake up briefly to check for the presence of a preamble signal, and do not periodically distribute common schedule information. Thus, these protocols exhibit a much lower power consumption with low traffic rates. WiseMAC applying preamble sampling and renouncing on costly synchronization schemes has a very low per-packet overhead, as it minimizes preambles by learning the neighboring nodes' schedules. The protocol thus manages to remain close to the ideal curve, but its power consumption and its throughput stalls at $\sim 35\%$ of that of CSMA. Interestingly, all protocols react to the linearly increasing and decreasing load level in a symmetric manner. The increase of power consumption during the load increase is symmetric to the decrease in power consumption during the load decrease, which is in general a desirable property.

6.2.4 The Energy-Throughput and Energy-Latency Trade-offs

All E^2-MAC protocols trade off Quality of Service versus higher energy-efficiency, and hence introduce higher delays and restrain the maximum achievable throughput in order to conserve energy. In this section, we examine these trade-offs with the simulated E^2-MAC protocols. By running each protocol with different parameter settings, we thoroughly investigated the behavior of each of the simulated E^2-MAC protocol mechanisms, and not just the behavior of one particular parameter choice. We refer to one parameter tuple per protocol as a *configuration* hereafter, e.g., one configuration for WiseMAC would be [Base Interval=200 ms, Duty-Cycle=1% (2 ms)], another one [Base Interval=500 ms, Duty-Cycle=0.4% (2 ms)].

Figure 6.9 and 6.10 illustrate the energy-throughput and energy-latency trade-offs of the simulated E^2-MAC protocols. Each dot represents the results of one particular protocol *configuration* in the simulation experiment outlined in Section 6.2.2. In Figure 6.9, the trade-off between maximum achieved throughput and energy-

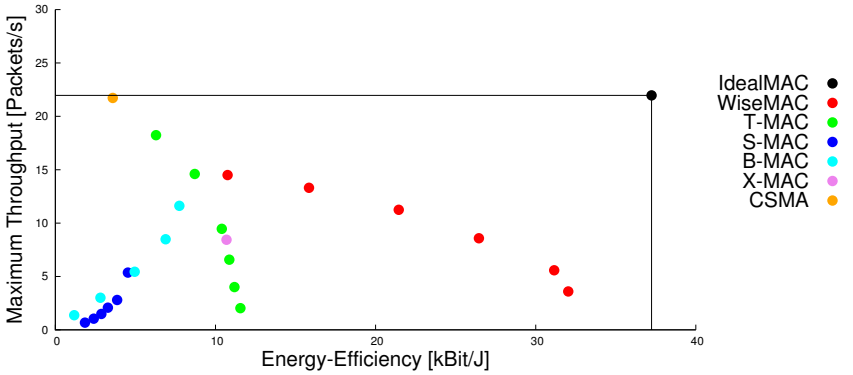


Fig. 6.9: Throughput vs. Energy-Efficiency

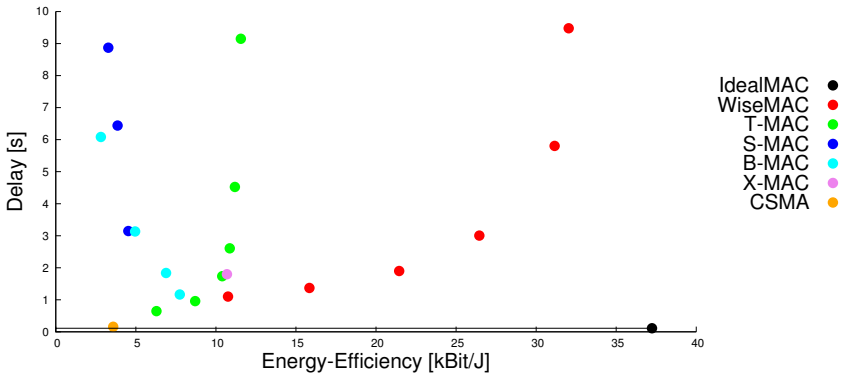


Fig. 6.10: Delay vs. Energy-Efficiency

efficiency of the simulated E^2-MAC protocols becomes very well visible. CSMA being energy-unconstrained has a very high maximum throughput. However, with CSMA not turning off the transceiver during the low-traffic phases, its energy-efficiency remains very low. The protocol efficiency is measured in in kBit/J, hence calculating how many *useful* (payload) bits have been transmitted from source to sink for each consumed Joule. *Ammer et al.* [9] have termed a similar concept as the energy-per-useful-bit (EPUB) metric. In contrast, we use the reciprocal coefficient in order to obtain a metric where *more is better*. The IdealMAC protocol, in which a receiver node always *knows* when to switch the transceiver to the receive mode to receive packets, has both a high throughput and a very high energy-efficiency. IdealMAC illustrates where the theoretic lower and upper bounds of the E^2-MAC protocol problems are - it is not possible to reach a higher throughput nor a higher efficiency than IdealMAC. No E^2-MAC protocol will ever get beyond the rectangle that is spanned by IdealMAC in Figure 6.9.

The different choices of the frame length and Base Interval parameter values visualize the trade offs between the two goals throughput and efficiency. When moving from the top leftmost dot towards the lower rightmost dot of the WiseMAC protocol, one can see how that the configurations with a higher achievable throughput come at the cost of a lower energy-efficiency: if WiseMAC is being operated with a long interval between two wake-ups, the protocol almost reaches the energy-efficiency of IdealMAC, but then only achieves a rather limited throughput. On the other hand, T-MAC can be tuned to reach almost the same throughput as CSMA, but at the cost of a decreasing energy-efficiency. X-MAC applying the wake-up interval adaptation algorithm reaches a fair throughput and tolerable delay at a reasonable efficiency, but its performance clearly lags behind that of WiseMAC. The main reason for this is the high per-packet overhead of the preamble strobes. One crucial advantage of X-MAC's strobed preamble mechanism is the possibility to let nodes adapt their sleep-wake interval to the traffic rate. Nodes with short wake-up intervals will respond earlier with an Early-ACK than nodes with a long wake-up interval. Hence, the protocol offers self-configuration and adaptation capabilities, while with other protocols, e.g., WiseMAC and T-MAC, the interval between two

wake-ups remains constant and does not adapt to the traffic rate.

Figure 6.10 similarly depicts the trade-off between the average delay and energy-efficiency. CSMA exhibits a low average delay at the cost of a low energy-efficiency. IdealMAC reaches both, a low delay at a very high energy-efficiency. IdealMAC again illustrates the lower bounds of the E^2-MAC protocol problem. While it is not possible to reach a higher throughput than IdealMAC, it is neither possible to reach a lower average delay. The energy-latency trade-off with the different *configurations* of T-MAC and WiseMAC becomes visible: when increasing the energy-efficiency of the protocol configurations by increasing the interval between two wake-ups, the delay accordingly increases, too. While T-MAC achieves a lower delay, WiseMAC exhibits a higher energy-efficiency.

6.3 Measuring Traffic Adaptivity

We investigated to find a means to *measure* and *quantify* the property of *traffic adaptivity* of an E^2-MAC protocol under varying traffic conditions, in order to compare the current state of the art in MAC protocol design and locate a starting point for investigations on further improvements. A traffic-adaptive E^2-MAC protocol should allow for using the radio-transceiver truly in an on-demand manner. It should use it as much as necessary to transmit and receive, whenever traffic needs to be handled, and turn it off when nothing has to be sent or received. We find that the question how well a protocol is able to adapt to varying traffic conditions can be rephrased by the question how the Quality of Service parameters throughput and delay behave under heavily varying traffic, and how energy-efficient the protocol remains under such conditions.

We developed a *tri-partite* metric that quantifies the ability of an E^2-MAC protocol to adapt to varying traffic conditions, based on a comparison with the IdealMAC reference protocol. The metric incorporates the energy-efficiency, the maximum achievable throughput, as well as the average delay, hence we refer to it as tri-partite. We briefly remind the mathematical properties a metric function $d(x, y)$ needs to fulfill: A metric d is a mapping $d : X \times X \rightarrow \mathbb{R}$ on any set X, with $\mathbb{R}$ being the set of real numbers. For all x, y, z in X, this function is required to satisfy the following conditions:

- $\mathrm{d}(x, y) \geq 0$ (non-negativity)
- $\mathrm{d}(x, y) = 0$ if and only if $x = y$ (identity of indiscernibles)
- $\mathrm{d}(x, y) = \mathrm{d}(y, x)$ (symmetry)
- $\mathrm{d}(x, z) \leq \mathrm{d}(x, y) + \mathrm{d}(y, z)$ (triangle inequality)

Our proposed metric for the *traffic adaptivity* of an E^2-MAC protocol under varying traffic measures the minimal *distance* between the different configurations of the protocol and the IdealMAC reference protocol. This distance is measured in the vector space spanned by the energy-efficiency (x), the maximum throughput (y) and the delay (z) measured in the above experiment of varying traffic. Hence,

we actually measure how much *worse* the protocol performs in comparison with IdealMAC. We represent the results of each configuration P_i of the simulated E^2-MAC protocol P as a tuple $(x_{p_i}, y_{p_i}, z_{p_i}) \in X \times Y \times Z$ where X is the energy-efficiency (measured in kBit/Joules), Y the maximum achievable throughput (packets/sec) and Z the average measured delay that the protocol exhibited in the varying load scenario. We further refer to $Id = (x_{Id}, y_{Id}, z_{Id})$ as the tuple representing the results of the IdealMAC protocol hereafter. The Euclidean distance d between IdealMAC and any configuration P_i of the E^2-MAC protocol denoted as:

$$d(P_i, Id) = \sqrt{(x_{p_i} - x_{Id})^2 + (y_{p_i} - y_{Id})^2 + (z_{p_i} - z_{Id})^2}$$

measures how much *worse* the configuration P_i behaves under the examined experiment conditions compared with IdealMAC. Figure 6.11 illustrates this difference between configurations of WiseMAC and T-MAC and the reference protocol IdealMAC (black vectors).

Applying the Euclidean metric to the measured values yields distances that are dependent of the scale of the axis. However, any metric assessing the adaptivity of E^2-MAC protocol should take the energy-efficiency, the maximum achievable throughput as well as the latency into account at equal ratios, and should not depend on the axis scale. A trivial change in the measurement units (e.g., kbps instead of bps) should not have an impact on the metric itself. Hence, we normalize the x, y and z-axis to take values in between the interval $[0, 1]$ and obtain the normalized distance d_{norm}:

$$d_{norm}(P_i, Id) = \sqrt{(\frac{x_{p_i} - x_{Id}}{x_{Id}})^2 + (\frac{y_{p_i} - y_{Id}}{y_{Id}})^2 + (\frac{z_{p_i} - z_{Id}}{z_{max} - z_{Id}})^2}$$

For the energy-efficiency and throughput, the upper bounds are determined by the efficiency of IdealMAC (x_{Id}) and the maximum throughput of IdealMAC (y_{Id}).The

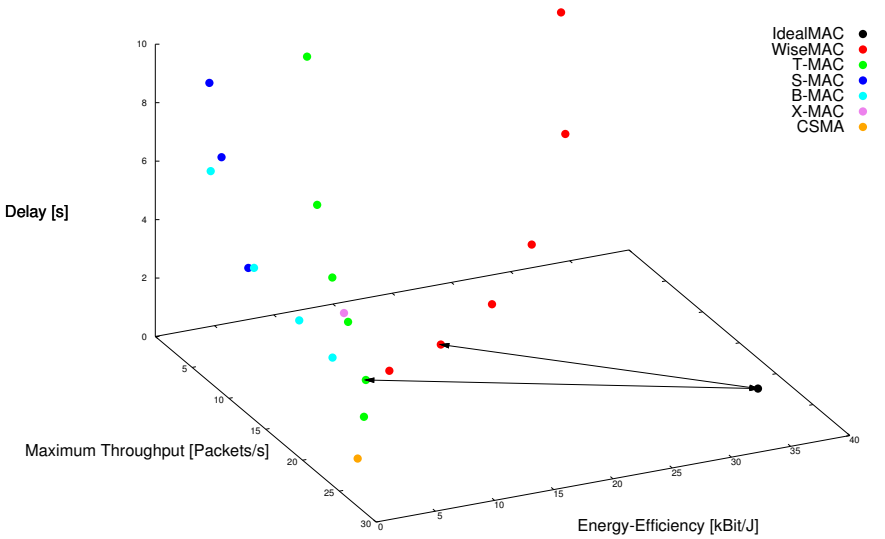


Fig. 6.11: Energy-Efficiency (x) vs. Maximum Throughput (y) vs. Delay (z)

value z_{max} corresponds to the *worst* measured delay of the simulated E^2-MAC protocols that does not exceed *10s*. Note that in contrast to the efficiency and the throughput, this particular choice of the maximum value for the delay has an influence on the metric itself.

We define the *traffic adaptivity* of every simulated E^2-MAC protocols as the distance of its *best* configuration P_i from IdealMAC. The best configuration of every E^2-MAC protocol P is its configuration $P_i = (x_{p_i}, y_{p_i}, z_{p_i})$ with the minimal distance to the IdealMAC reference protocol. The traffic adaptivity TA of a protocol P denoted as a set of its configurations $P_0, P_1, \cdots, P_k$ then yields as:

$$TA(P) = \min d_{norm}(P_i, Id) \;\; (P_i \in P)$$

Obviously, the TA-metric based on the normalized Euclidean distance fulfills the mathematical properties of a metric function. IdealMAC describes the best possible behavior of a MAC protocol with respect to adaptability under variable traffic. Figures 6.7 and 6.5 illustrate how an optimally traffic-adaptive MAC protocol should behave, allocating as much resources as necessary to handle the imposed traffic, and saving as much as possible. We hence consider the distance from a MAC protocol to IdealMAC to be the best characterization for its ability to deliver Quality of Service *and* high energy-efficiency under variable load.

Applying the Metric to today's E^2-MAC protocols

Applying this metric to simulated E^2-MAC protocol yields the results depicted in Figure 6.12. The figure depicts the protocols' TA-value measured in the above-mentioned experiment on the y axis. With taking the normalized distance function d_{norm}, the TA value is in between [0, $\sqrt{3}$] for every protocol.

WiseMAC [57] proved to be the protocol offering the best *traffic adaptivity* of the simulated set of protocols. The preamble sampling mechanism and the learning of the neighboring nodes' schedules leads to a high energy-efficiency, and choosing suitable values for the basic listen interval leads to reasonable multi-hop delays. *Demirkol et al.* [41] pointed out that WiseMAC performed better than S-MAC "under variable traffic conditions", and that its base mechanism exhibits a good

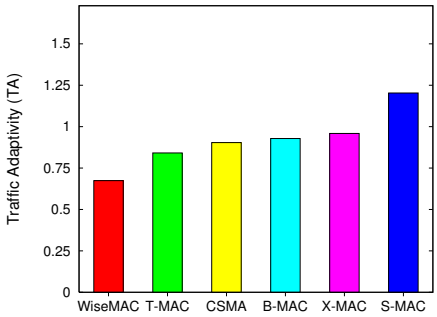


Fig. 6.12: Traffic Adaptivity Metric TA

"adaptivity to changes" - a conjecture for which we have just delivered a proof. *Langendoen et al.* [107] point out that WiseMAC shows a "remarkable consistent behavior across a wide range of operational conditions, always achieving the best or second-best performance". However, as the WiseMAC protocol does not yet integrate any self-configuration and self-adaptation mechanisms, the protocol performance is very much dependent of choosing suitable parameter settings for any given scenario. WiseMAC restrains the maximum achievable throughput quite heavily, compared to energy-unconstrained CSMA.

The protocol T-MAC [38] has proven to achieve the highest throughput at load peaks and a low latency. Its *traffic adaptivity* measured using the TA-metric depicted in Figure 6.12 is lower, since the TA-value is roughly 20% above that of WiseMAC. The T-MAC time-out mechanism that prolongs the T-MAC duty-cycle noticeably pays off, as T-MAC achieves a much higher throughput than its predecessor S-MAC, and - given a short frame length and contention window - comes closest to the throughput of CSMA. The drawback then consists in the decreasing energy-efficiency, since T-MAC does not apply similarly brief duty-cycles as WiseMAC, but keeps the radio on for at least the duration of the contention period and a CTS frame in every interval.

Weighted Metric

There is an infinite number of mappings between any three-dimensional space $X \times Y \times Z$ and the real numbers $\mathbb{R}$. For certain scenarios and applications, it might make sense to define a metric where throughput and latency is *more important* than the energy-efficiency and thus has a larger weight. The metric proposed in this section can easily be adapted for such purposes. One can redefine $d_{norm}(P_i, Id)$ and introduce weight factors $\omega_x, \omega_y, \omega_z \in (0, \infty)$, which account for the importance of the energy-efficiency, throughput and delay, and hence obtain the weighted metric $d_{norm}(P_i, Id, \omega_x, \omega_y, \omega_z)$ defined as:

$$d_{norm}(i, j, \omega_x, \omega_y, \omega_z) = \sqrt{\frac{\omega_x(\frac{x_{p_i}-x_{Id}}{x_{Id}})^2 + \omega_y(\frac{y_{p_i}-y_{Id}}{y_{Id}})^2 + \omega_z(\frac{z_{p_i}-z_{Id}}{z_{max}-z_{Id}})^2}{\omega_x + \omega_y + \omega_z}}$$

Setting different values for the weight factors $\omega_x, \omega_y, \omega_z$ hence yields different orderings of the examined protocols. Dividing through the sum of the weight factors $\omega_x, \omega_y, \omega_z$ maps the applied metric back to an interval between [0, 1] for each protocol. Hence, the only thing that matters is the ratio between the weights - e.g., weight factors $\omega_x = \omega_y = \omega_z = 1$ map to the same value as weight factors $\omega_x = \omega_y = \omega_z = 3$. Figures 6.13 and 6.14 illustrate the weighted metric for values of $\omega_x = 1, \omega_y = 1, \omega_z = 1$ and $\omega_x = 1, \omega_y = 3, \omega_z = 3$, respectively. One instantly notices that with the second set of values, the ordering of the protocols changed significantly. When assigning a higher importance to throughput and latency, the protocols T-MAC and CSMA conveying a higher maximum throughput and lower latencies overtake the WiseMAC protocol.

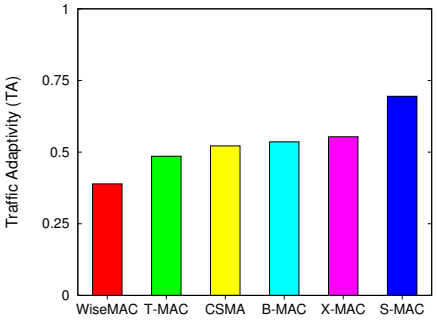


Fig. 6.13: Traffic Adaptivity Metric $\omega_x = 1, \omega_y = 1, \omega_z = 1$

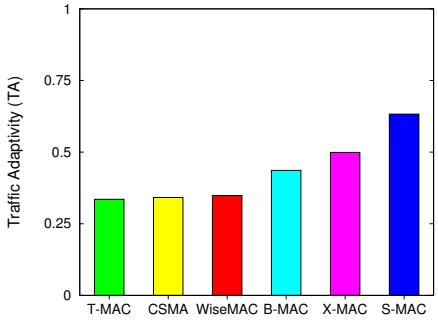


Fig. 6.14: Traffic Adaptivity Metric $\omega_x = 1, \omega_y = 3, \omega_z = 3$

6.4 Conclusions

In this chapter we have explored the design space of today's most frequently cited E^2-MAC protocols, the current *state of the art* in MAC traffic conditions. We have defined a metric to quantify the ability of a protocol to operate under variable traffic load. The metric compares a MAC protocol's performance with respect to throughput, latency and energy-efficiency against an idealized concept of an E^2-MAC protocol named IdealMAC. The metric shows how far a E^2-MAC protocol is from being able to truly allocate the radio transceiver in an *on-demand* manner. Many of todays E^2-MAC protocols exhibit a very high energy-efficiency, some of them yet come close to the theoretic lower bounds of IdealMAC. This gain in efficiency, however, comes at the cost of severely restrained maximum throughput, as well as massively increasing end-to-end packet latency.

With the observations summarized above, we motivate our contributions and our investigations on traffic-adaptive MAC mechanisms in the subsequent Chapter 7. A general-purpose E^2-MAC protocol with a high energy-efficiency under low traffic, which is capable to adapt its behavior in case of higher traffic is definitely yet missing. Such a protocol should ideally exhibit an energy-footprint such as WiseMAC, but offer stable Quality of Service under higher load. Another important observation and conclusion we draw from the evaluation in this chapter is that the choice of essential protocol parameters has shown to have a vast impact on the resulting protocol characteristics, in particular the ratio between energy-efficiency and throughput. Intelligent design of mechanisms that alternate between different tuples of protocol parameters is therefore a promising option to achieve higher run-time traffic adaptivity on the MAC layer. We follow exactly this strategy in Chapter 7, and design a MAC protocol that bases on a combination of WiseMAC and X-MAC for the low/idle traffic case. WiseMAC has shown in this chapter to come close to the theoretic lower bounds of energy-efficiency in the low/idle traffic case, but suffers from severely restrained maximum throughput and sharply increasing latency under higher load (cf. Figure 6.11), which disqualifies it for a wide range of throughput- and latency-sensitive applications with temporally high traffic, e.g., classical event-based alarming applications or WMSNs, cf.[4][7].

Chapter 7

The Maximally Traffic-Adaptive MAC (MaxMAC) Protocol

In this chapter, we present our contributions towards flexible and traffic-adaptive medium access control mechanisms in wireless sensor networks, a gap we have observed to be yet missing in our initial evaluation of the WiseMAC burst transfer mode in Chapter 5 and in particular in the analysis of the six different MAC protocols in Chapter 6. With the introduction of the *Maximally Traffic-Adaptive MAC (MaxMAC)* protocol in [85] and [88], we have targeted at designing MAC mechanisms that achieve two goals: to support *energy-efficient operation* at times of sparse network activity, but provide measures to operate with sound Quality of Service at times of increased traffic load. We have integrated established design principles for MAC protocols developed over the last decade with novel run-time adaptation techniques to effectively allocate the radio truly in an *on demand* manner: turning it on when traffic has to be handled, keeping it off when not.

The chapter is organized as follows: Section 7.1 motivates our contribution and relates them to the work presented in Chapters 6 and 5. Section 7.2 describes the basic design and operation principles of the MaxMAC protocol. Section 7.3 discusses particularities of the network simulation environment, which are basically the same as in Chapter 5. Section 7.3.2 then discusses simulation results of the MaxMAC protocol and relates them to those of a selection of reference protocols. The chapter continues with discussing the functionality of the first prototype implementation of MaxMAC in Section 7.4. Sections 7.4.3 and 7.4.4 discuss measurement results of our MaxMAC prototype implementation obtained in a series of distributed real-world experiments. Section 7.6 concludes the chapter and gives an outlook on future work and promising features that yet remain to be examined.

7.1 Motivation

In Chapter 6, we have observed that today's most widespread Energy-Efficient MAC (E^2-MAC) protocols generally reduce the energy consumption spent for the radio transceiver at the cost of deteriorating Quality of Service, in particular by an

increase of the packet latencies and a sometimes massive decrease of the achievable throughput and reliability. In the MAC protocol domain, researchers have concentrated almost exclusively on the energy aspect, often leaving aside any Quality of Service considerations. The maximum throughput of today's E^2-MAC protocols is limited to only a fraction of that of energy-unconstrained MAC protocols.

Tight restrictions with respect to throughput and latency may be tolerable in many WSN applications, especially when the main focus of the operated network consists in monitoring a large area or an object of interest with a few measurement samples per day or per hour. However, besides such classical environmental monitoring applications, numerous rather event-based applications have recently emerged, which have more demanding requirements with respect to Quality of Service parameters during certain operation periods. Event detection, localization, tracking and alarming applications require good Quality of Service during short periods of increased network activity, as well as a high efficiency during longer periods of inactivity in order to prolong network operability to more than a few days.

Concrete examples for such use cases can be found, e.g., in monitoring systems for health-care (cf. CodeBlue [119] and/or *Chipara et al.* [35]), or in disaster-aid and first-response systems, cf. *Gao et al.* [67]. Potential use cases can definitely be found in the broad area of event-based environmental monitoring systems. Wherever sensor nodes need to monitor environmental values, but have to be able to instantly react when certain events have been recognized (e.g., the volcano monitoring [183] or forest fire prevention system [98]), Quality of Service can become an important issue. *Wittenberg et al.* [186] develops a collaborative event detection system built to monitor fenced areas and prevent intrusion, which requires fast response times when an event is being sensed. Varying, temporarily high traffic can further be expected in the emerging field of Wireless Multimedia Sensor Networks (WMSNs) [4][7], e.g., when from time to time, a rather large image has to be transmitted. Once an event has been detected, a *traffic-adaptive* MAC protocol's primary objective should shift from saving energy towards delivering good Quality of Service (high throughput, low delay). Reasonable Quality of Service provided *on demand* facilitates manual real-time interaction of network operators with WSNs. In such scenarios, Chapter 6 has clearly conveyed that most E^2-MAC protocols do not provide reasonable flexibility, as they fail to exploit the capacity of the wireless channel and massively increase end-to-end latencies compared to energy-unconstrained MAC protocols.

7.2 MaxMAC Design

Since the first appearance of the E^2-MAC protocol S-MAC in 2002 [190], enormous progress has been made in this field, especially in decreasing the idle duty-cycle. The idle duty-cycles have come down from 100% with energy-unconstrained CSMA, over 10% with the first E^2-MAC protocols (S-MAC [190], T-MAC [38]) to less than 1% in today's state of the art preamble-sampling based approaches

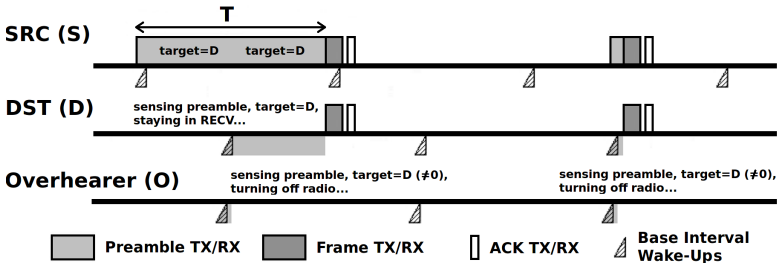


Fig. 7.1: Preamble Sampling with Embedded Target Address in MaxMAC

(e.g., B-MAC [143], WiseMAC [57] or X-MAC [25]). The reduction of the duty-cycles from 100% to less than 1% has boosted lifetimes of sensor nodes equipped with a pair of AA batteries by an order of two magnitudes, while still preserving network connectivity. With respect to minimizing the idle power consumption and duty-cycle of the radio, *Langendoen* [109] concludes in its survey study on MAC protocols that there is probably not much more room for improvement.

7.2.1 Basic Media Access Mechanism

MaxMAC takes advantage of the substantial work carried out on E^2-MAC protocols in the last decade, especially the asynchronous contention-based protocols B-MAC [143], WiseMAC [57] and X-MAC [25]. This section briefly discusses the basic media access mechanisms used in MaxMAC, while Section 7.2.2 discusses its run-time traffic adaptation and dynamic resource allocation mechanisms.

Preamble Sampling

With Preamble Sampling (also referred-to as Low-power-Listening) introduced in B-MAC and WiseMAC, nodes keep their radios off for most of the time and only wake up for brief periodic duty-cycles to poll the channel for a preamble signal once every *Base Interval* T (cf. Figure 7.1). We chose the WiseMAC preamble sampling and pairwise schedule learning mechanism as entry point and *default behavior* for MaxMAC due to the resulting low per-packet overhead and low idle power consumption. The sound performance and high efficiency of WiseMAC under various conditions has been independently pointed out in [107] and [85][90], before and after MaxMAC's initial appearance in [85] (cf. Chapter 6). Among the preamble-sampling-based E^2-MAC protocols, it must be seen as the most efficient approach, since it only employs a minimal preamble for collision avoidance and clock drift compensation, as opposed to B-MAC or X-MAC, which send out long preambles before every payload frame transmission. Unsurprisingly, the recently developed ContikiMAC [48], which integrates concepts from several E^2-MAC protocols, also applies the WiseMAC preamble minimization to reduce the transmission overhead. Section 2.4.3 of Chapter 2 discusses the different preamble sampling techniques of B-MAC and WiseMAC in detail.

Overhearing Avoidance

The preamble sampling technique of WiseMAC is already quite efficient in avoiding costly overhearing. With sparse traffic, chances are high that the wake-ups of non-targeted receivers do not coincide with those of the targeted receivers. However, with higher traffic and transmissions of queued packet trains, overhearing of preambles and frames can also become an increasing source of energy waste. MaxMAC minimizes overhearing by enriching preambles with target ID information, as illustrated in Figure 7.1. Target nodes turn their radio transceiver on, sense the carrier for *their particular preamble* to receive preamble and frame. Non-target nodes turn their radios on, extract the target information in the ongoing preamble transmission, notice that they are not targeted and immediately turn the radio off again. This concept has been applied in X-MAC [25], where nodes send preamble strobes in between which receiver nodes can signal reception readiness with a so-called *Early-ACK*. MaxMAC is the first protocol that merges this concept of integrating a target address identifier into the preamble in order to reduce overhearing with the highly efficient preamble minimization technique of WiseMAC.

7.2.2 Run-Time Traffic Adaptation Mechanisms

In contrast to most of today's E^2-MAC protocols, which operate with rather static parameter settings, MaxMAC introduces traffic-adaptation features to instantly react to changing load conditions by altering its behavior at run-time. Similarly as in dynamic frequency/voltage scaling, where the CPU reacts to higher computation load with an increase of the frequency/voltage, a traffic-adaptive E^2-MAC protocol should react to changing load by correspondingly tuning the radio: turning it on more frequently when more traffic has to be handled, keeping it permanently on during load peaks, and turning it off again when the load level permits it.

Allocation/Deallocation of Extra Wake-Ups

With E^2-MAC protocols alternating between statically configured sleep in each interval, given that no traffic adaptation mechanisms are integrated. Latency typically increases sharply, as forwarding nodes need to buffer incoming frames and wait for the next wake-up of their intermediate gateway node, which often sums up to several seconds in multi-hop scenarios. The first traffic adaptation feature and essential novelty of MaxMAC tackles this very decisive E^2-MAC protocol drawback. In MaxMAC, nodes change their state (and hence, their behavior) and allocate so-called *Extra Wake-Ups* when the rate of incoming packets reaches predefined threshold values, and de-allocate them when the traffic rate drops below these thresholds again, falling back to their initial channel sampling behavior. Figure 7.2 illustrates the state-based adaptivity mechanism with a source node (SRC) sending packets to a receiver node (DST) with increasing rate. Nodes operate in the *Base Interval* state per default, polling the channel periodically each Base Interval T. They alter their state (and behavior) by switching to states S_1, S_2 when

the corresponding thresholds T_1, T_2 are reached. Thresholds T_1 and T_2 are set to 2 and 6 packets/s in the illustration in Figure 7.2, but only serve to illustrate the basic concept - different values will be chosen in the subsequent evaluations in Sections 7.4.3 and 7.4.4. Each node keeps estimating the rate of incoming packets, using a sliding window of 1s (cf. rate estimation graph of DST in Figure 7.2).

With the rate of incoming packets reaching the threshold T_1, the DST schedules one additional *Extra Wake-Up* in between each Base Interval, effectively doubling the amount of duty-cycles over time. The receiver node DST communicates its increased wake-up frequency in the ACK. SRC receives this announcement and marks the increased wake-up frequency of node DST in its schedule offset table. With the notification sent by DST in the ACK, DST *promises* to remain in the new state and keep its increased wake-up frequency for a predefined timespan S1_LEASE. For each state in MaxMAC, the LEASE timespans (S1_LEASE, S2_LEASE, CSMA_LEASE) define how long a node *promises* to remain in the new state when announcing the state change in the ACK. LEASE timespans can be *extended* in any new ACK transmission. By remaining in a higher state for at least the LEASE duration, fast oscillation between the different states can be mitigated.

With the rate of incoming packets reaching the threshold T_2, DST changes to state S_2, doubles the amount of wake-ups again and announces its state change in the ACK (cf. Figure 7.2). As soon as these timespans expire, nodes having received prior state change announcements will assume that the corresponding node has fallen back to its default behavior, polling the channel with the Base Interval T, which prevents them from transmitting when the target is not awake.

As observed in Chapter 6, increasing the amount of wake-ups is an effective, yet considerably cheap means of increasing network throughput and decreasing end-to-end latency. If SRC needs to forward other packets, the time to wait for the next wake-up of DST is halved with DST being in state S_1 or even quartered with DST being in state S_2. If the additional wake-ups scheduled by DST are not used for transmissions, the waste of energy remains limited, as some few additional channel polls are energetically inexpensive, as they are de-allocated after the LEASEs.

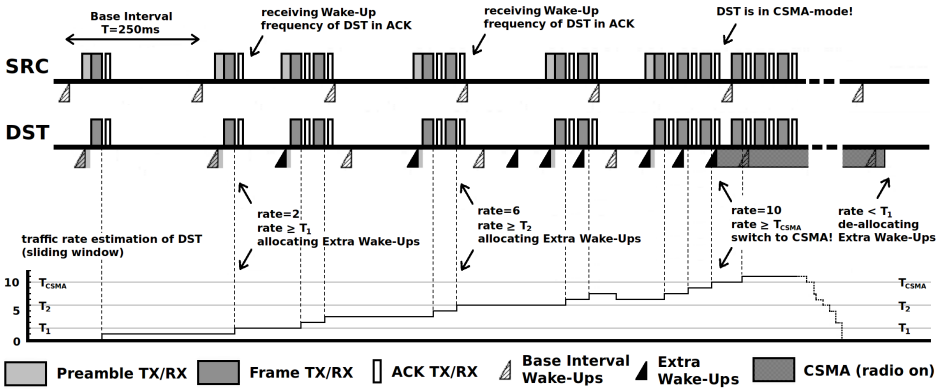


Fig. 7.2: Rate Estimation, Extra Wake-Ups and CSMA mode in MaxMAC

Exploiting the Channel Capacity by switching to CSMA

Most E^2-MAC protocols have been designed under the assumption of sparse low-rate traffic, and hence take into purchase a severe degradation of the maximum throughput compared to non duty-cycled MACs. As discussed in Chapter 6 and likewise in *El-Hoiydi et al.* [56] or the T-MAC study [38], they only reach a fraction of that of CSMA. MaxMAC has been specifically designed to achieve a throughput similar as CSMA in situations of increased network activity, which can be seen as *best case* for the class of contention-based random-access MAC protocols. While the allocation of *Extra Wake-Ups* helps to achieve a somewhat increased throughput and reduces the latency, the characteristics of CSMA can still not be reached. MaxMAC thus carries the concept of changing the *behavior* one step further: when the rate of incoming packets reaches a further threshold T_{CSMA} (with $T_{CSMA} > T_2 > T_1$), MaxMAC switches to energy-unconstrained CSMA and announces this state change to the sender node (and potentially overhearing nodes) in the ACK. Figure 7.2 illustrates node DST measuring the rate of incoming packets to reach $T_{CSMA} = 10$ packets/s in the right part of the figure. DST hence switches to the CSMA state, announcing the state change to SRC in the ACK, hence *promising* to remain in the CSMA state for at least the predefined timespan CSMA_LEASE. Within this timespan, SRC can transmit packets without having to wait for a wake-up of DST, as it *knows* that DST keeps its transceiver on for at least the timespan CSMA_LEASE. With CSMA_LEASE expiring, all nodes having received the prior state change announcement of DST assume that DST has fallen back to the Base Interval state, which prevents them from transmitting at times when DST is asleep.

Figure 7.3 illustrates the state-based adaptivity concept of MaxMAC with the state transitions as a finite state machine. Nodes switch from the Base Interval state to a higher state $S_1, S_2, CSMA$ when the rate reaches the associated thresholds T_1, T_2, T_{CSMA}. When switching from the Base Interval state to S_1 or S_2, nodes schedule *Extra Wake-Ups* and double or quadruple their wake-up frequency. When the rate reaches the threshold T_{CSMA}, nodes switch to energy-unconstrained CSMA and keep their radio transceivers turned on. With the traffic load falling below T_{CSMA} and the timeout CSMA_LEASE expiring, nodes switch again to states S_1

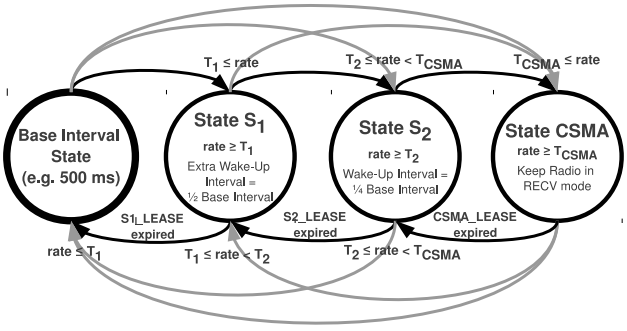


Fig. 7.3: Finite State Machine-based Traffic Adaptivity Concept of MaxMAC

or S_2 and restart alternating between brief channel polls and long sleep intervals. Nodes completely de-allocate all *Extra Wake-Ups* and fall back to the Base Interval state when the packet rate drops below T_1 and all LEASE timespans have expired. The MaxMAC traffic adaptation mechanism scales well for multi-hop topologies, as each node measures and reacts upon a given rate increase in a decentralized manner. State changes are communicated efficiently in the data frame header and the ACK frames, neglecting the need to introduce new costly control messages.

Threshold Values and Parameters

We have illustrated the MaxMAC adaptivity concept with four states so far in this chapter. However, the number of intermediate states between the *Base Interval State* and the *CSMA* state can generally be chosen arbitrarily, and would typically depend on the transmission rate of the radio transceiver of the used target platform. The thresholds and LEASE parameters allow for fine-tuning the MaxMAC protocol and its properties. Choosing low values for the thresholds makes sense in delay-sensitive applications, since the thresholds would quickly be exceeded and the protocol would quickly allocate more resources. Higher values make sense in rather energy-sensitive and delay-tolerant applications. The threshold values of the subsequent Section 7.3.2 were chosen according to the results of unmodified WiseMAC with different wake-up frequencies of Chapter 6.

In an attempt to augment the MaxMAC concept to become more generally applicable, we explored the feasibility of self-configuration approaches for finding optimal values for the threshold parameters at run-time. However, after preliminary simulations, we came to the conclusion that integrating such mechanisms would raise more questions and problems than they would solve. First, one has to face the fact that the threshold settings have an immediate impact on the trade-off between the resulting Quality of Service and energy-efficiency. Per definition, there is never an *optimum* in any trade-off, since the improvement of one design goal naturally leads to depreciation of the other. Second, a run-time calibration of the threshold settings would require a network-wide parameter negotiation scheme. Since MaxMAC relies on *promises* of the wake-up frequencies, nodes need to operate with the same parameters as their neighbors or at least *know* what parameters the neighboring nodes apply, in order to be able to predict their next wake-ups. Parameter changes would then impose the need for *telling* the neighboring nodes what parameters were just chosen, which would massively impact on the communication overhead. We hence decided to choose the different sets of parameters, between which MaxMAC switches at run-time, in an offline manner at compile-time, as done in basically every other E^2-MAC protocol. A distributed approach where Quality of Service target goals are defined by the network operator, which are then translated by the nodes in the network to individual parameter sets using a distributed algorithm, would be certainly the most elegant solution. To design and evaluate such an algorithm, however, seemed too ambitious before any real-world prototype had yet verified that the initial protocol design fulfilled our expectations.

7.3 Simulation-based Evaluation

In order to investigate the effectiveness of our proposed traffic adaptation mechanisms, we implemented MaxMAC in the same OMNeT++[178] environment as outlined in Chapter 6. We intended to compare the MaxMAC simulation model to our small library of MAC protocol models in OMNeT++ in a series of simulation experiments before actually prototyping it on a real-world platform, an approach that is often pursued in the WSN community today.

Section 7.3.1 lists the parameters of the OMNeT++ simulation environment. Sections 7.3.2 and 7.3.3 evaluate the MaxMAC simulation model in two different simulation scenarios, and compare it against a selection of the most frequently cited E^2-MAC protocols. Section 7.3.3 takes up the notion of the tree-partite metric defined in Chapter 6 to classify MaxMAC in the design space of the simulated E^2-MAC protocols and examines its advantages and disadvantages.

7.3.1 Simulation Model and Parameters

Radio Transceiver and Energy Consumption Model: We modeled the radio transceiver with a finite state machine model consisting in the states sleep, receive and transmit, weighted with the respective energy costs, the same model that is usually applied in network-simulator based evaluations of WSN MAC protocols, e.g., [38][72][107]. Although we have shown in Chapter 4 that taking into account the transceiver switches into the energy model increases the accuracy, we consider the abovementioned *Three States Model*'s degree of realism sufficient for preliminary simulator-based evaluations. The MSB430 platform [14], which we used for our real-world prototype implementation of MaxMAC, is equipped with the CC1020 radio [173]. Table 7.1 lists its current, voltage and transmission rate. Unfortunately, the applied transmission rate of 115'200 bps later turned out not to be supported by the employed ScatterWeb^2 Operating System, which is why we had to use the default transmission rate of the ScatterWeb^2 OS of 19'200 bps instead.

MAC Protocol Simulation Models: We compared MaxMAC against the E^2-MAC protocols S-MAC, T-MAC, B-MAC, WiseMAC and X-MAC, as well as the reference protocols IdealMAC and energy-unconstrained CSMA, the same models as described in Chapter 6. Table 7.2 again lists their main parameters. As the protocol behavior often heavily depends on the choice of the essential protocol parameters (e.g., Base Interval, Duty-Cycle), we studied the protocols with different *configurations* of those parameters, by varying the parameters over a wide range, and not just one particular parameter choice. The examined protocol configurations are essentially the same as those described in the evaluations of Chapter 6, except for MaxMAC, which was introduced in this chapter.

In order to speed up computations, we used the University of Bern UBELIX High-Performance Computing Cluster [36] to let the simulation runs be executed in parallel on the many available CPU cores.

Transceiver CC1020 [173]:			
Transmit Current I_{tx}	21.9 mA	Supply Voltage U	3 V
Recv Current I_{rx}	17.6 mA	Transmission Rate R	115'200 bps
Sleep Current I_{sleep}	1 μA		
OMNeT++ Parameters:			
Path Loss Coefficient α	3.5	Transmit Power	0.1 mW
Log-normal Deviation σ	2.5 db	SNR Threshold	4 dB
Carrier Frequency	868 MHz	Sensitivity	-100.67 dBm
Carrier Sense Sensitivity	-112 dBm		
Simulation Settings:			
Simulation Runs	100	Simulated Time	1000 s
Maximum Retries	3	Frame Header Size	14 bytes
Payload	50 bytes		

Table 7.1: Simulation Model and Experiment Parameters

CSMA	Contention Window CW	10 ms
S-MAC	Listen Interval	[100, 200, 300, 500, 1000, 2000] ms
	Duty-Cycle	10%, 20%
T-MAC	Frame Length	[50, 100, 200, 300, 500, 1000] ms
	Contention Window CW	5 ms
	SYNC size D_{SYNC}	14 bytes
	RTS size D_{RTS}	14 bytes
	CTS size D_{CTS}	10 bytes
	Timeout	$1.5 \times (CW + D_{RTS}/R + D_{CTS}/R)$
	SYNC period	10 s
B-MAC	Base Interval	[25, 50, 100, 200, 500] ms
	Duty-Cycle	[8, 4, 2, 1, 0.4] %
	Medium Reservation Interval	uniform (0,10) $\times$ t_{rx-tx}
WiseMAC	Base Interval	[25, 50, 100, 200, 500, 1000] ms
	Duty-Cycle	[8, 4, 2, 1, 0.4, 0.2] %
	Medium Reservation Interval	uniform (0,10) $\times$ t_{rx-tx}
X-MAC	Max Interval	[100, 200, 500] ms
	Min Interval	10 ms
	Early-ACK size D_{EACK}	10 bytes
	Inter-Strobe-Interval	D_{EACK}/R
	Listen Interval	$D_{EACK}/R + t_{rx-tx} + t_{tx-rx}$
MaxMAC	Base Interval	[50, 100, 200] ms
	Duty-Cycle (idle)	4, 2, 1%
	S1_LEASE	1 s
	S2_LEASE	1 s
	CSMA_LEASE	1 s
	T_1	4 Packets/s
	T_2	8 Packets/s
	T_{CSMA}	12 Packets/s

Table 7.2: E^2-MAC Protocol Parameters

7.3.2 Traffic along a Multi-Hop Chain

We evaluated two different scenarios with the selected E^2-MAC protocols and our proposed MaxMAC protocol. The first simulation scenario consists in a simple chain of nodes, exactly the same scenario setup as evaluated in Chapter 6. The second scenario discussed in Section 7.3.3 models an event-based scenario in a larger-scale grid network of 49 nodes.

In the chain scenario, the source node A located on one end of the chain generates traffic, which is then forwarded hop-by-hop towards the sink node H on the other end. A similar scenario setup is chosen in the studies on S-MAC [190] and B-MAC [143]. Again, we varied the offered traffic from low rates to high rates during each run, as our major interest is the protocol adaptivity at *run-time*. Figure 7.4 displays the offered load generated at the application layer of the source node. The load is low (0.1 packets/s) for most of the time, but there are peaks where the packet rate is increased, up to a maximum rate of 22 packets/s. We again chose 22 packets/s as the maximum as this had proved to be the maximum throughput that CSMA could handle without major packet loss. When increasing the rate above this rate, throughput stalls and additional packets are dropped due to buffer overflows or are lost due to collisions. Numerically, the slopes for the first four and the subsequent four load peaks amount to $\pm$ 0.16 packets/s^2 and $\pm$ 3.0 packets/s^2 for the respective load increase and decrease periods, respectively.

Throughput and Power Consumption

Figure 7.5 displays the rate of received packets at the sink node versus simulation time. The curves are averaged from 100 simulation runs for each protocol. As one can clearly see comparing the received packets in Figure 7.5 with the offered load in Figure 7.4, IdealMAC manages to handle all packets from source to sink. CSMA only suffers minor packet loss at the load peaks. The throughput of WiseMAC and T-MAC stalls at maximum 8 packets/s and 9 packets/s, respectively, which corresponds to roughly 35-40% of that of CSMA. Figure 7.5 clearly shows that MaxMAC with its state-based run-time traffic adaptation mechanism reaches the same throughput as energy-unconstrained CSMA. As the protocol allocates more duty-cycles or even totally switches to CSMA-like behavior at high traffic rates, the protocol manages to handle the load peaks without major packet loss.

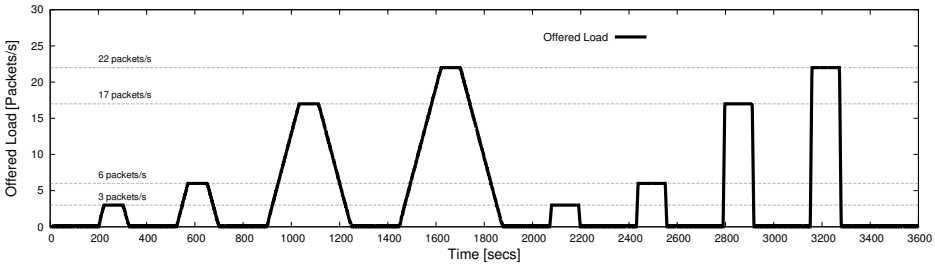


Fig. 7.4: Offered Load (Packets/s)

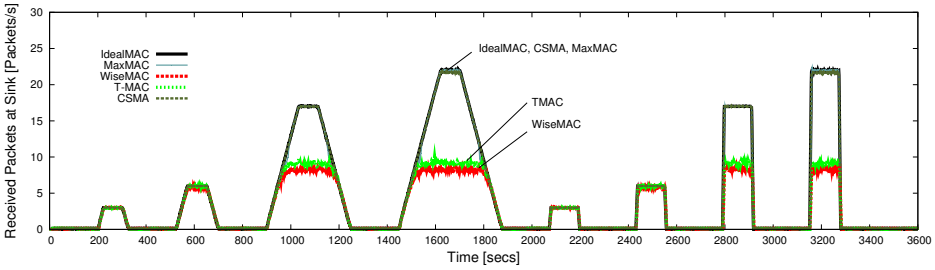


Fig. 7.5: Throughput at Sink

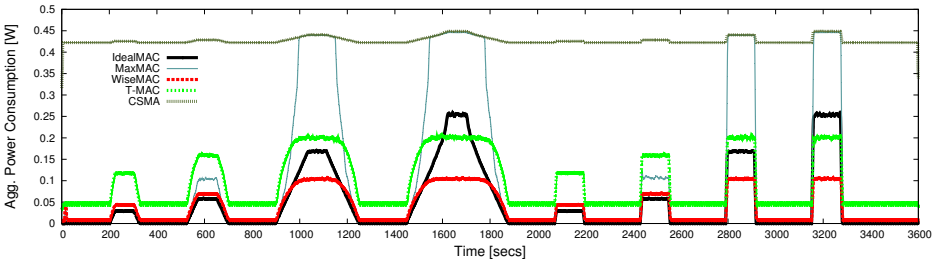


Fig. 7.6: Aggregated Network Power Consumption

Figure 7.6 depicts the aggregated power consumption of all eight sensor nodes' radio interfaces versus simulation time. One can clearly see the big gap between the E^2-MAC protocols and energy-unconstrained CSMA. With low traffic, CSMA wastes a lot of energy on idle listening. The load peaks are hardly visible at all, as the transceiver does not require much more power when transmitting, compared to idle listening, cf. the CC1020's radio specs [173]. The IdealMAC reference protocol illustrates the ideal behavior of an E^2-MAC protocol, allocating as much energy as needed to handle the imposed load, and immediately deallocating it with decreasing load. WiseMAC renouncing on costly synchronization schemes has a low per-packet overhead, minimizing preambles by learning adjacent nodes' schedules. It exhibits a low power consumption during the low traffic phases. Its throughput, however, stalls at roughly 35% of that of CSMA. T-MAC achieves a slightly higher throughput, but its idle power consumption is above that of WiseMAC, mainly due to the SYNC overhead to keep the nodes' wake-ups synchronized.

Thanks to the run-time traffic adaptivity mechanisms of MaxMAC, it reaches the same energy-efficiency in the low-traffic-phases as WiseMAC, but is able to handle the load peaks with much lower packet loss. As MaxMAC switches to the CSMA-state with the rate reaching $T_{CSMA} = 12$ packets/s (cf. Table 2), the power consumption of MaxMAC accordingly jumps to the level of CSMA at this rate, too. Figure 7.6 further illustrates that the *on-demand* resource allocation scheme of MaxMAC further succeeds astonishingly well when the packet rate decreases. With traffic rates decreasing towards 0.1 packets/s after the load peaks, MaxMAC quickly falls back to the states S_2 and S_1 and finally the Base Interval state, where it again exhibits a very low energy footprint, since it only samples the channel once in each Base Interval.

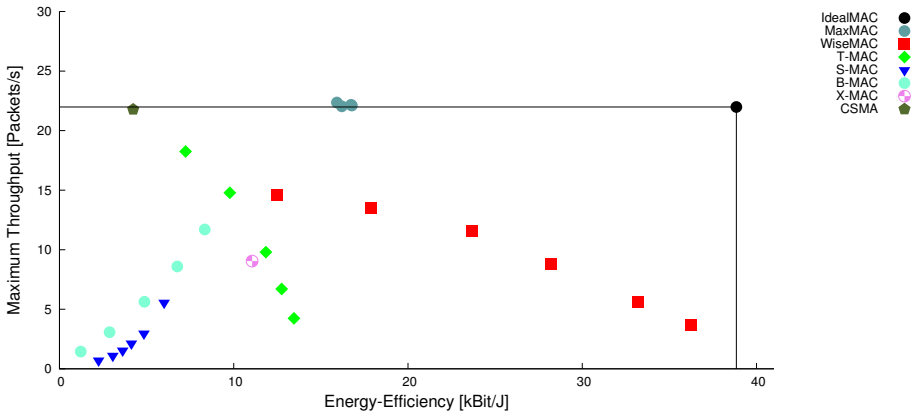


Fig. 7.7: Throughput vs. Energy-Efficiency

Energy-Throughput and Energy-Latency Trade-offs

We thoroughly examined the different MaxMAC protocol configurations with respect to the energy-throughput and energy-latency trade-offs. Figures 7.7 and 7.8 illustrate the measured trade-offs in the aforementioned experiment and compares MaxMAC with the E^2-MAC protocols discussed in Chapter 6. Each dot represents the results of one particular protocol *configuration* in the node chain experiment. In Figure 7.7, the trade-off between maximum achieved throughput and energy-efficiency of the simulated E^2-MAC protocols becomes well visible. The protocol's energy-efficiency is measured in in kbit/J, hence calculating how many *useful* (payload) bits have been transmitted from source to sink for each consumed Joule, exactly as done in the evaluation of Chapter 6. CSMA obviously achieves a high maximum throughput, its energy-efficiency, however, remains very low.

The rectangle spanned by IdealMAC in Figures 7.7 and 7.8 again illustrates the lower bounds of the E^2-MAC protocol problem: while it is not possible to reach a higher throughput or a higher efficiency coefficient than IdealMAC, it is neither possible to reach a lower delay. WiseMAC with its short channel polls achieves a high energy-efficiency, however, at the cost of a massively restrained maximum throughput and increasing end-to-end latency. Thanks to its run-time traffic adaptation mechanisms, MaxMAC reaches the same throughput as energy-unconstrained CSMA, but exhibits a much higher energy-efficiency. Although MaxMAC switches to CSMA-like behavior in the high traffic phases, its efficiency coefficient is higher than that of most of today's E^2-MAC protocols. The advantage of achieving the high throughput of CSMA *and* a much better energy-efficiency than most E^2-MAC approaches is a clear novelty in the design space of today's E^2-MAC protocols. Figure 7.8 similarly depicts the trade-off between average packet delay and energy-efficiency. Thanks to the scheduling of *Extra Wake-Ups*, which reduces the interval between two wake-ups, and the switch to CSMA-like behavior at even higher rates, MaxMAC reaches a far lower average end-to-end latency as other E^2-MAC protocols. MaxMAC achieves a delay which is - given

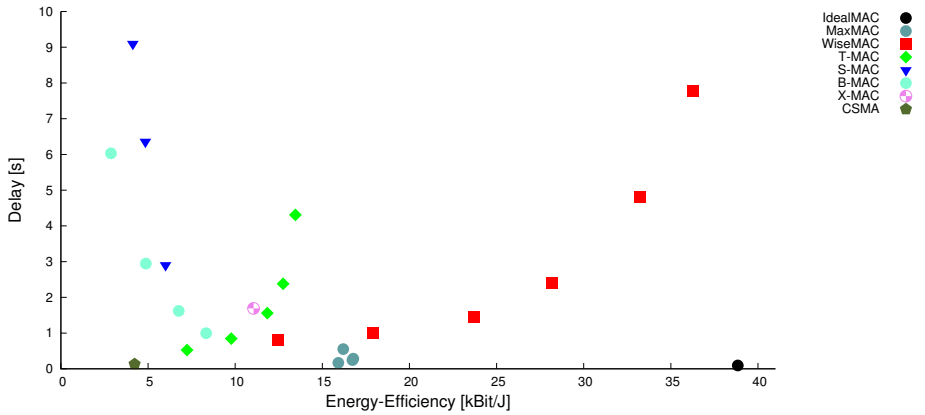


Fig. 7.8: Delay vs. Energy-Efficiency

the best examined configuration - only 70% higher than that of CSMA (compared to some 1000% with other E^2-MAC protocols), but achieves an energy-efficiency that is more than three times better than that of of CSMA.

Figure 7.9 represents the results of each configuration of the simulated E^2-MAC protocols as a tuple in the vector space $X \times Y \times Z$ where X is the energy-efficiency, Y the maximum achievable throughput (packets/s) and Z the average measured delay - in analogy to Section 6.3 of Chapter 6 - illustrating the design space and the potential for optimization in current E^2-MAC protocols. In Chapter 6, we have concluded that most protocols are not sufficiently adaptive, as they do not alter their behavior with respect to the load conditions. Although there is sufficient channel capacity, most existing protocols still turn their radio transceivers off too aggressively. MaxMAC is clearly distinguishable from the examined reference protocols

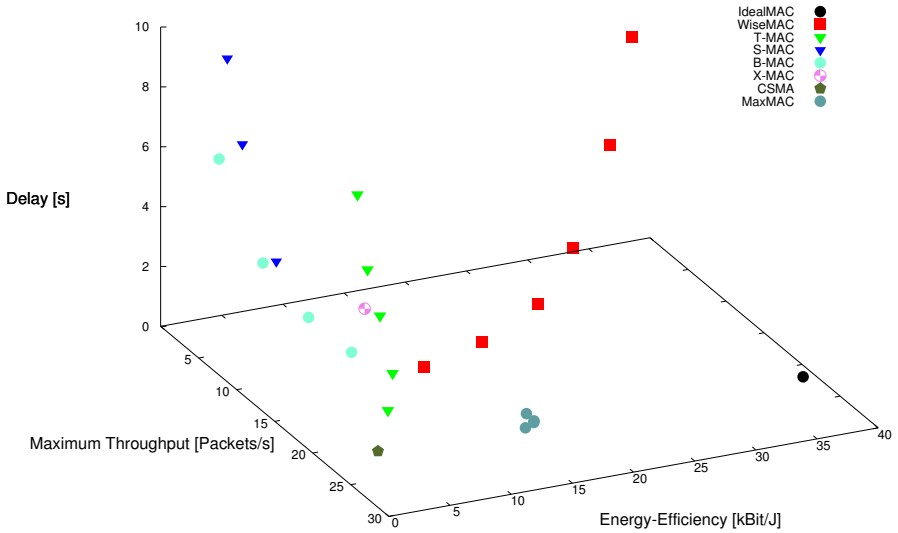


Fig. 7.9: Energy-Efficiency (x) vs. Maximum Throughput (y) vs. Delay (z)

by its ability to reach the same throughput and a similarly low latency as energy-unconstrained CSMA, while still exhibiting a good energy-efficiency during the considerably long periods of sparse network activity. The three examined *configurations* of MaxMAC hence exhibit the shortest distance to the IdealMAC protocol in the lower right corner in Figure 7.9, due to the high throughput, low delay and good energy-efficiency measured in the experiment.

7.3.3 Random Correlated Event Traffic

With our second experiment we examine the behavior of MaxMAC and the reference protocols in a larger scenario with a *correlated event workload* model, as proposed by *Hull et al.* [82]. We simulate a 49-node grid network (7x7) with the center node forming the sink, exactly the same scenario setup as applied by *Yanjun et al.* [188], which also use the RCE model. The distance between two adjacent nodes is 30m. With our parameters of the Log-Normal channel model [153], packet error rates are 1% and 15% on a straight link (30m) and a diagonal link (42.42m).

We apply a simple event traffic model that mimics the effects of spatially correlated events, in the same manner as in [82] and [188]. Spatially-correlated events are expected to occur in many event-based scenarios for WSNs, e.g., in the beforementioned monitoring applications in health-care systems, disaster-aid systems or tracking applications. The traffic model picks a uniform random (x,y) location for each event. Every node within the event sensing range R of this location then reports data packets with a rate of r_{event} during t_{event} towards the sink. We chose values of $R = 30m$, $r_{event} = 6$ packets/s and $t_{event} = 10s$ for the events being triggered each $30s$ at a uniformly distributed random location (x,y) of the network.

In large event-based scenarios (e.g., a monitoring application), the packet delivery rate (PDR) is usually given higher priority than the throughput per second. We hence measured the packet delivery rate, the average source-to-sink packet delay and the energy-efficiency (in terms of kBit/J) during 100 runs of $3600s$. Packets

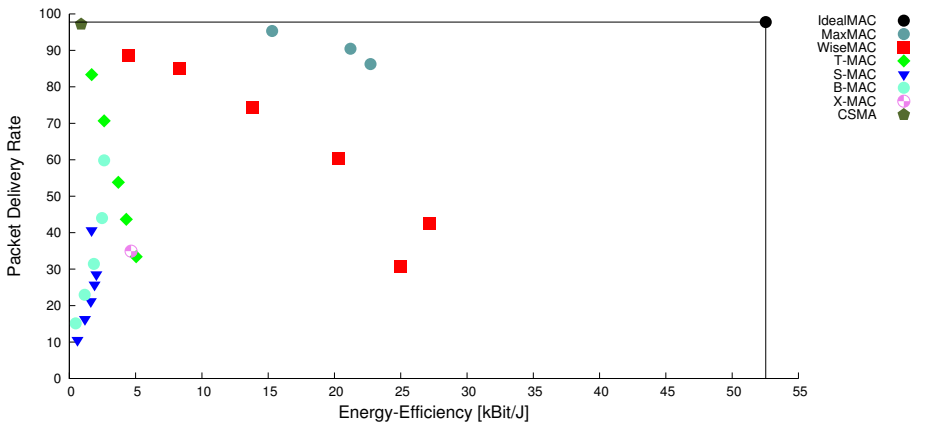


Fig. 7.10: 49-Nodes Grid Scenario: Packet Delivery Rate (PDR) vs. Energy-Efficiency

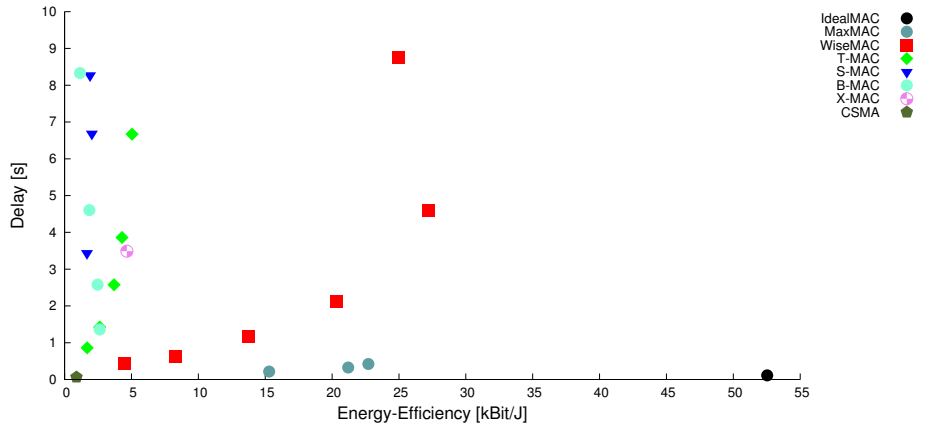


Fig. 7.11: 49-Nodes Grid Scenario: Delay vs. Energy-Efficiency

are routed along the *shortest path*. Nodes select their parent node randomly in the initiation phase of the experiment if there are multiple nodes exhibiting the same hop count. Figure 7.10 depicts the packet delivery rate (PDR) vs. energy-efficiency of the different *configurations* of the E^2-MAC protocols in the random correlated event experiment. Energy-unconstrained CSMA and IdealMAC reach a PDR of almost 100%. Some packets are lost due to buffer overflows, as the transmit buffer is assumed to be limited to 10 packets. Thanks to its run-time adaptation mechanisms, MaxMAC reaches a similar PDR as CSMA, while still exhibiting a much higher energy-efficiency. Although the protocol switches to CSMA in the high traffic phases, its overall efficiency is still higher than that of most other E^2-MAC protocols. The combination of a high PDR *and* energy-efficiency achieved by MaxMAC's adaptation mechanisms is well-visible in Figure 7.10 and constitutes a clear benefit and novelty compared to the other E^2-MAC protocols.

Figure 7.11 depicts the trade-off between average source-to-sink packet delay and energy-efficiency in the random correlated event experiment. Thanks to the scheduling of *Extra Wake-Ups* and switching to CSMA at higher rates, the three examined *configurations* of MaxMAC reach a far lower average source-to-sink latency as all the other E^2-MAC protocols. The adaptivity concept of MaxMAC further fits to the event-based traffic: with an event being triggered at a random location, nodes start reporting data along the shortest path to the sink. With the load reaching the MaxMAC thresholds T_1, T_2, T_{CSMA}, nodes alter their behavior in order to deliver the pending load. After the event has been processed and the packet stream ends, the LEASE timespans time out and MaxMAC again falls back to the default behavior in the Base Interval state.

A drawback of the MaxMAC adaptivity concept is the fact that the protocol requires a certain adaptation time, during which the adaptation mechanisms are triggered. In multi-hop scenarios, all nodes forming a route from the event source to the sink first need to reach the given thresholds. During this *adaptation phase*, packets are lost mainly due to buffer overflows, as the PDR in Figure 7.10 exhibits.

Thereafter the traffic adaptation strategy achieves a high throughput and low average delay. We explored a scheme where the application layer triggers an initial packet that *paves the way* for upcoming traffic. In such a packet, the application layer would specify information regarding the amount of data which needs to be transmitted. In case of a large bulk data transfer (e.g., a low-resolution image, cf. the scenario in the subsequent Section 7.4.4), such a single initial packet could then signal to all the nodes to remain in the CSMA state until completion of the transfer. In Chapter 9, we will indeed show that this approach leads to a better net efficiency than keeping the nodes duty-cycling the radio. However, we finally omitted to further investigate such a scheme, since we wanted to avoid a tight coupling of the involved communication protocol layers and wanted to move on with prototyping MaxMAC on real-world sensor nodes.

Measuring Traffic Adaptivity

In order to fully close the gap between the state of the art analysis of the most widely known E^2-MAC protocols conducted in Chapter 6, and the evaluation of MaxMAC in this chapter, we apply our proposed metric for traffic-adaptive behavior defined in Chapter 6 to the results of this section. The Traffic Adaptivity metric $TA(P)$ given in equation (TA) measures the traffic adaptivity of a protocol by computing the *minimum* distance between the performance of its evaluated *configurations* (the tuples forming the set $P = \{P_1, \ldots P_k\}$) and that of IdealMAC.

$$d_{norm}(i, j, \omega_x, \omega_y, \omega_z) = \sqrt{\frac{\omega_x(\frac{x_{p_i} - x_{Id}}{x_{Id}})^2 + \omega_y(\frac{y_{p_i} - y_{Id}}{y_{Id}})^2 + \omega_z(\frac{z_{p_i} - z_{Id}}{z_{max} - z_{Id}})^2}{\omega_x + \omega_y + \omega_z}}$$

$$TA(P) = \min d_{norm}(P_i, Id, \omega_x, \omega_y, \omega_z) \;\; (P_i \in P) \qquad \text{(TA)}$$

In order to *measure* the impact of the adaptation mechanisms, we applied the TA-metric to all the results of the *Random Correlated Event* experiment described and evaluated in Section 7.3.3. This yields the TA-values depicted in Figures 7.12

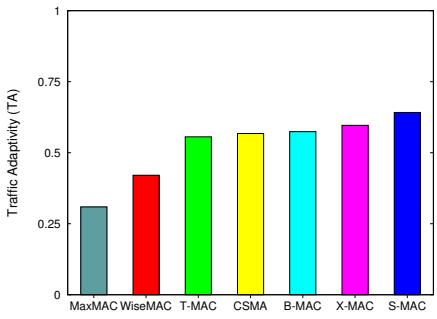


Fig. 7.12: Traffic Adaptivity Metric TA
$\omega_x = 1, \omega_y = 1, \omega_z = 1$

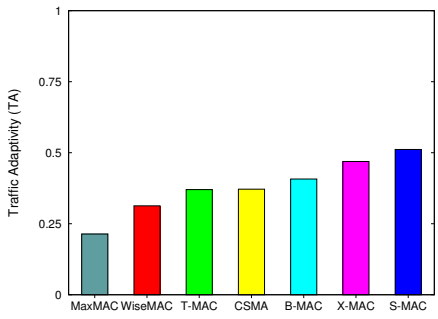


Fig. 7.13: Traffic Adaptivity Metric TA
$\omega_x = 1, \omega_y = 3, \omega_z = 3$

and 7.13. One can clearly see that the property of adaptivity as defined in Chapter 6 has been improved significantly with the introduction of MaxMAC. The TA-value of WiseMAC, which used to be the best-performing protocol in Chapter 6, could be reduced by roughly 30%. The two figures depict the normalized distance with different ratios for the weight factors (cf. Section 6.3 of Chapter 6). The impact of the improvement does slightly change with altering the weight factors. However, when assigning a three times higher importance to both the maximum throughput and the latency (cf. Figure 7.13), the improvement of MaxMAC over the other protocols still remains in the same range (27.2% and 32.7% for the different weight factor sets).

Rethinking the Traffic Adaptivity Metric

The metric defined in equation (TA) calculates how far the *best configuration* of an examined E^2-MAC protocol is from IdealMAC. The IdealMAC protocol is a concept that demonstrates how an optimally traffic-adaptive MAC protocol should ideally behave, allocating as much resources as necessary to handle the imposed traffic, and saving as much as possible. However, we have observed that one major *aspect* of traffic-adaptivity is not reflected with the metric in equation (TA). In Figures 7.9, 7.10 or 7.11, it is clearly observable that the three configurations of MaxMAC convey almost the same results with respect to the measured maximum throughput, latency and energy-efficiency. The ability of the protocol to converge towards the same consistent behavior with different parameter settings should also be taken into account in a traffic adaptivity metric. MaxMAC configured with the initial Base Intervals of 50 ms, 100 ms and 200 ms converged to almost the same behavior, compared to e.g. WiseMAC with the same Base Interval parameters.

The ability to converge towards a consistent behavior with similar Quality of Service parameters is wiped away in the metric defined with TA by taking the *minimum distance* of the different results of the different protocol configurations. However, the sheer distance between the different protocol configurations themselves, without taking into account the distance to IdealMAC, can neither be considered as a meaningful metric. We hence redefined the traffic adaptivity metric TA to TA_{mod}, which averages the distance to IdealMAC of each MAC protocol configuration, instead of calculating the minimum distance.

$$TA_{mod}(P) = \sum_{i=0}^{k} \frac{d_{norm}(P_i, Id, \omega_x, \omega_y, \omega_z)}{k} \quad (P_i \in P) \qquad (TA_{mod})$$

By averaging the distances to IdealMAC in the three-dimensional space, the metric penalizes those protocols that convey heavily differing results for different parameters sets. Figures 7.14 and 7.15 depict the modified traffic-adaptivity metric TA_{mod} with the same weight factors and simulation results as Figures 7.12 and 7.13. Again, the improvement of MaxMAC over the other protocols remains in the same range for the different weight factor sets. However, the improvement

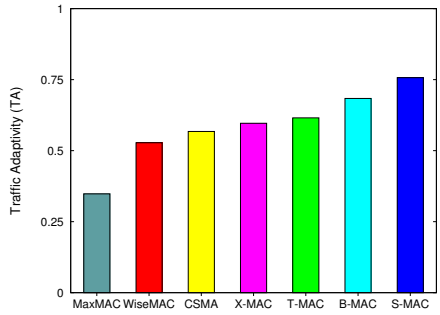


Fig. 7.14: Traffic Adaptivity Metric TA_{mod} $\omega_x = 1, \omega_y = 1, \omega_z = 1$

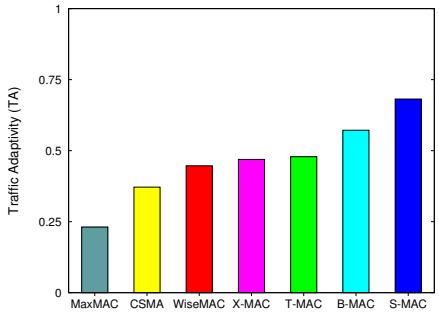


Fig. 7.15: Traffic Adaptivity Metric TA_{mod} $\omega_x = 1, \omega_y = 3, \omega_z = 3$

over the next-best protocol is even higher. The TA_{mod} value of MaxMAC is 35.5% and 37.9% lower than that of the next better MAC protocol. The downside of the modified metric TA_{mod} is the fact that TA_{mod} depends much more from the choice of the parameter sets, with which the protocols are examined. While a very bad configuration has no impact in the initial definition (TA), since only the minimum distance is computed, a configuration leading to very bad results can massively deteriorate the modified metric TA_{mod} (TA_{mod}).

Conclusion

Although we have managed to describe the problem of traffic-adaptivity in the three dimensional space spanned by the throughput, latency and energy-efficiency, finding a formal metric that delivers an unambiguous notion has proven to be harder than expected.

The application of the initial TA-metric to MaxMAC specified in (TA), and the modified TA-metric specified in (TA_{mod}) are an attempt to *formally describe* and *quantify* the improvement in traffic adaptivity achieved with MaxMAC. Clearly, the MaxMAC protocol is unlike most other E^2-MAC protocols able to find a fair balance between energy-efficiency under low or mediocre traffic load, while still being able to achieve the maximum throughput that can be expected for contention-based random access MAC protocols. This property must be seen as a clear novelty in the design space of the evaluated E^2-MAC protocols. The simulation results of this section hence confirm our expectations that the MaxMAC concept effectively succeeds in combining the advantages of energy unconstrained CSMA (high throughput, high PDR, low latency) with those of classical E^2-MAC protocols (high energy-efficiency).

7.4 Prototype-based Evaluation

Inspired by the sound results of MaxMAC in the simulation environment, we went on to implement a prototype of the protocol on a real-world sensor network platform. We chose the MSB430 [14] sensor node platform using the ScatterWeb^2 Operating System [157]. The MSB430 platform has a CC1020 [173] byte-level radio transceiver operating in the 804-940 MHz ISM frequency band. We specifically chose this platform due to its byte-level oriented transceiver chip, since preamble-sampling mechanisms can generally be easier implemented on byte-level radios than packet-oriented radios such as the CC2420 [174].

We first implemented the WiseMAC [57] protocol to form a starting point for our implementation and investigation of the MaxMAC concept in real-world environments. In the subsequent evaluations of this section, we compare our MaxMAC protocol to WiseMAC, and to a slightly altered version of the default IEEE 802.11-like CSMA MAC protocol in ScatterWeb^2 OS [14], which does not duty-cycle the radio in any form, and avoids collision using a random-backoff mechanism.

While the maximum raw bit rate of the CC1020 is 153.6 kbit/s, the ScatterWeb^2 OS currently only supports a data rate of 19.2 kbit/s. The simulation results can hence only partly be compared to the results of the real-world implementation.

Table 7.3 lists the packet and header format used for all of the MAC protocol implementations. Table 7.4 then lists the main MAC protocol parameters. The upper half in Table 7.4 lists general settings that apply for all examined MAC protocols, as well as the common WiseMAC and MaxMAC parameters. In the lower half, the table lists the MaxMAC-specific parameters, e.g., the state thresholds or the LEASE timeouts. The Base Interval T with which nodes sample the channel was set to 500 ms, the time for a channel poll to 3 ms. We experimentally determined that the MSB430 sensor node with its CC1020 [173] radio driver requires roughly 3 ms to turn the radio on and to reliably determine if the preamble byte is being received. The *duty-cycle*, calculated as the fraction of the time the radio is kept on

Field	*Bytes*	*Description*
Preamble	variable	predefined bit sequence (0xAA)
Start Delimiter	3	indicates the beginning of the data
Size	1	packet size, including payload
Address Target	1	address of the receiver (0 - 254)
Flags	1	MaxMAC flags (e.g., state info)
Address Source	1	address of the sender (0 - 254)
Number	1	packet sequence number
Type/More Bit	1	packet type, containing more-bit
Millis	1	milliseconds until next wake-up
Payload	28 bytes	payload data
CRC	2	CRC-16 checksum

Table 7.3: MaxMAC Prototype Packet Format

General Parameters	
Bitrate	19'200 bps
Baudrate	38'400 bps
Packet size	40 byte (incl. header)
Packet Queue size	7 packets
WiseMAC & MaxMAC Parameters	
Base Interval T	500 ms
Duty-Cycle	0.6 % (3 ms)
MaxMAC-specific Parameters	
Threshold T_1	1 packet/s
Threshold T_2	2 packets/s
Threshold T_{CSMA}	3 packets/s
S1_LEASE	3 s
S2_LEASE	3 s
CSMA_LEASE	3 s

Table 7.4: MAC Protocol Parameters

(receive / transmit) during each Base Interval T hence amounts to 0.6 %, given the node is idle and neither receives nor transmits any packets. The threshold parameters in Table 7.4 have been experimentally determined in a small scale scenario, e.g., the parameter T_{CSMA}=3 packets/s was configured according to the observation that the throughput of WiseMAC stalls at a rate of 3 packets/s across 2 hops (cf. Figure 7.21 in the next section).

In order to allow for a fair comparison, we implemented a packet burst mode for each MAC protocol, such that nodes can transmit queued packet trains in a burst. Nodes can signal pending packets to the receiver using a so-called *More Bit* in the header, and continue transmitting packets in a burst, receiving an acknowledgment for each frame, exactly as in the simulation models of Section 7.3.2.

In case a sender does not receive the respective acknowledgement after a frame transmission, the retry mechanism is invoked. The number of retransmissions for each frame is set to 1 for all protocols. In CSMA, the sender again contends for the medium and - given the channel is sensed free - transmits again. In WiseMAC, the node attempts again at the subsequent next wake-up. In MaxMAC, the retransmission mechanism depends on the adaptivity state of the receiver, waiting for the next wake-up in case of the base interval, S_1 and S_2, and immediately attempting to retransmit in case the receiver is in the CSMA state.

7.4.1 Run-time Traffic Adaptivity in the MaxMAC Prototype

This section describes the implementation of the run-time adaptivity features of MaxMAC in the real-world prototype. According to the state-based concept depicted in Figures 7.3 and 7.2 of Section 7.2, MaxMAC allocates so-called *Extra Wake-Ups* when the rate of incoming packets reaches the predefined threshold values. In that way, MaxMAC stepwise shifts from the objective of energy conservation towards the objective of providing higher Quality of Service, which is

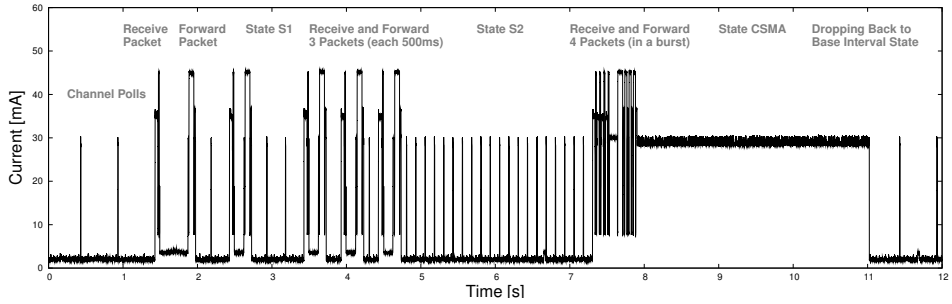


Fig. 7.16: Node forwarding Packets in different MaxMAC States

basically driven by the *demand* of the layers above the MAC. The threshold values T_1, T_2, T_{CSMA} listed in Table 7.4 were chosen to suit for the targeted behavior of our MaxMAC prototype on the MSB430 sensor node platform. As in any MAC protocols, there are no globally *optimal* values. Manipulating the threshold values allows the network operator to fine-tune the MaxMAC protocol to reach its targeted behavior and performance. Choosing low values would particularly make sense in delay-sensitive applications, since the thresholds would be quickly exceeded at the price of a higher energy footprint. Section 7.3.2 elaborates in more detail on threshold values and parameters.

Figure 7.16 displays an excerpt from the current trace (sampled at 1000 Hz) of a node running MaxMAC and receiving and forwarding packets with increasing rate, using the SNMD devices, which are discussed in detail in Section 2.3.2 of Chapter 2. Two packets are received and subsequently forwarded at t=1.5s and t=2.5s. Further three packets are received and forwarded at t=3.5s, t=4s, t=4.5s and a burst of 4 packets in a burst is received and forwarded starting at t=7.25s. The node measures the received rate of packets using a sliding window over 1 second. It operates in the *Base Interval* state per default, polling the channel each 500 ms, and switches to the states S_1, S_2 when the corresponding load thresholds T_1, T_2 are reached. With the rate of incoming packets reaching the threshold T_1, the node switches to the state S_1 and schedules one additional *Extra Wake-Up* in between each Base Interval. This state change is well visible in Figure 7.16 after t=2s, after the node has forwarded the received packet. With exceeding T_2 (between t=4s and t=5s), it switches to the state S_2 and again doubles its wake-up frequency. Increasing the amount of wake-ups is an effective, yet energetically cheap means of increasing network throughput and decreasing end-to-end latency. The decrease in latency is well visible in Figure 7.16: the time difference between the reception of the first packet and its transmission to the next node is much shorter at higher rates. Packets can be forwarded faster with the receiver being in state S_1, S_2, or even CSMA, as the time gaps between packet receptions and the receiving nodes' next wake-ups decreases, which massively reduces the end-to-end packet latencies.

MaxMAC has been specifically designed to maximize throughput in situations of increased network activity, however, still sticking to the contention-based random access nature. The scheduling of *Extra Wake-Ups* somewhat increases the achiev-

able throughput, but still it remains much below that of CSMA. Hence, when the rate of incoming packets reaches the threshold T_{CSMA}, MaxMAC switches to the CSMA state, where it completely abandons any sleep-wake pattern. The node in Figure 7.16 switches to the CSMA state at t=7.5s after the reception of the packet burst. With the LEASE timeouts expiring, MaxMAC falls back to its default behavior, where nodes poll the channel with the Base Interval T. The fallback mechanism is well visible in Figure 7.16 at t=11s where the node leaves the CSMA state and falls back to the *Base Interval* state, sampling the channel every 500 ms.

7.4.2 Implementation Pitfalls

When implementing the MaxMAC concept on the MSB430 sensor node platform, several challenges and pitfalls have been faced, which we did not anticipate during our prior investigations based on the OMNeT++ network simulator. This section briefly describes some of the encountered issues and - where it was possible - our workarounds or countermeasures to cope with the encountered difficulties.

Inaccuracies of the Software-based Timers

The ScatterWeb^2 operating system offers to schedule functions to *Timers*, which are then executed after a specified interval. However, the execution of the timers unfortunately comes with some inaccuracies, since the CPU is handling timers and events one after the other in a linear manner in the operating system's *superloop* (cf. Section 2.1 of Chapter 2). When certain events currently use the CPU, e.g., because a received packet has to be read from the incoming receive buffer and passed to the application, or because some other timer is currently using the CPU, a ScatterWeb^2 timer might be executed later than specified. This is generally not a big problem, since most WSN applications usually do not use the CPU for intensive computations, including our traffic generators used in this chapter. However, the inaccuracies turned out to play a crucial role in the MaxMAC and WiseMAC prototypes, because these protocols rely on very exact timings of the receiver's wake-ups. The shortened preambles do not span across an entire wake-up Base Interval, as e.g, in X-MAC or B-MAC, but are transmitted just before the receiver's advertised wake-up, and hence require more precise timing. When simply scheduling a timer each 500 ms to implement the duty-cycling, the timer inaccuracies would lead to the node's wake-ups drifting apart by several milliseconds within only few tens of seconds, rendering the WiseMAC schedule offset table useless.

We addressed this problem by scheduling the timers slightly before the advertised wake-up times, and then waiting in a busy-wait loop running the `NOP` instruction for some few milliseconds and checking the current time in each loop, in order to precisely turn the transceiver on at the wake-up time. For the transmission, the timing issue was less of a problem, since the entire transmission and reception mechanism is handled within designated interrupt service routines, which are executed prioritized and which do not rely on the same sort of timers.

Embedding Target ID into Preamble

In the first ScatterWeb v.1.0 WiseMAC prototype on the ESB nodes, which is discussed in Chapters 2 and 5, the radio transceiver TR1001 [156] applied a synchronization mechanism that assured correct interpretation of the byte bounds while receiving data. The overhearing avoidance mechanism discussed in Section 7.2 could hence be accomplished by repetitively transmitting the node identifier as preamble byte, and letting receiver nodes only listen to frames that are announced with their identifier as preamble byte.

With the CC1020 [173] transceiver, however, the received byte bounds are not automatically determined. The ScatterWeb^2 operating system therefore integrates a rather complex mechanism that finds the beginning of a frame and the correct byte bounds based on a series of frame delimiter bytes, which are received after a variable number of 0xAA (or 0x55) symbols (= an alternating sequence of '1' or '0'). In order to implement a preamble sampling mechanism with a minimal idle duty-cycle, we prolonged this byte sequence to represent the preamble tone. A receiver hence only has to turn on its radio transceiver, listen whether the 0xAA (or 0x55) byte is being received, and can turn off the radio if this is not the case. However, we failed to integrate the node identifier due to the missing synchronization of the byte bounds. We did attempt to alter the preamble byte sequence to integrate the node identifier after a series of 0xAA (or 0x55) bytes, but found that the full integration of the proposed ID embedding mechanism would significantly increase the idle duty-cycle. Finally, we chose to keep the lower idle duty-cycle, which in the current WiseMAC and MaxMAC implementation only accounts to 0.6%, and omitted the integration of the overhearing avoidance mechanism proposed in Section 7.2.

Carrier Sensing Range

El-Hoiydi et al. propose in [57][56] to apply an *Extended Carrier Sensing Range* of roughly two times the transmission range, in order to not only avoid collisions, but also the effects of the *hidden node* problem. Figure 7.17 depicts this concept: when node C is transmitting a packet to D, the nodes within its carrier sensing range (A, B, D, E) should all remain silent and defer pending transmissions in order not to interfere with the receiver of C's transmission. We have implemented this mechanism in the simulation model of WiseMAC and MaxMAC, but had to

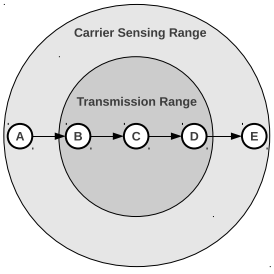


Fig. 7.17: Extended Carrier Sensing Range in WiseMAC [57]

omit it in our real-world prototype for several reasons. First, the recognition of the medium occupancy on the CC1020 relies on the measured RSSI value. This value is, according to the radio specs datasheet [173], compared against a "programmable threshold" and saved to a certain register. Due to insufficient knowledge of the internal operation of this transceiver, we failed in finding a way to alter this "programmable threshold". Second, the digital RSSI value of the CC1020 turned out to vary heavily among the different node instances, rendering the definition of RSSI thresholds impossible. Since we are rarely dealing with circular transmission ranges in real-world, we omitted to further investigate the *Extended Carrier Sensing Range* mechanism, which has anyway only been evaluated in simulation in [57].

7.4.3 Tabletop Experiments

In this section, we evaluate our prototype implementation of the MaxMAC protocol in two small-scale experiments conducted on top of an office table. We compare it against the implementation of unmodified WiseMAC [57] and the ScatterWeb^2 operating system's energy-unconstrained IEEE 802.11-like CSMA variant. The setup of the experiment on top of a table allows us to conveniently measure the current draw of one node with an SNMD and run a series of controlled and repeatable experiments without unpredictable interferences or reflection effects. The results further allow for an in-depth analysis of the current draw and throughput over time.

Three Nodes Chain Scenario

The first experimental results of the MaxMAC prototype were gained using three nodes *A, B* and *C* aligned on a table with a distance of 50 cm between them, hence with all nodes being in each other's transmission range (cf. Figure 7.18). We let node *A* generate traffic of variable rate towards node *B* forwarding it to node *C*, according to the load curve depicted in Figure 7.20. The load alternates between no traffic and load peaks of increasing intensity, ranging from 0.5 packets/s to 6 packets/s. Figure 7.21 depicts the rate of received packets at the sink node *C*, filtered with a Central Moving Average Filter of 1s and averaged across 20 experiment runs. During the entire experiment duration, node *B* was attached to a SNMD device and the node's current draw was sampled at 1000 Hz. Before every measurement run, an external node was used to broadcast a SYNC frame to all nodes on the table. Immediately upon reception of this frame, all nodes on the table reset their DCO clocks back to zero. Since this is done inside the interrupt service

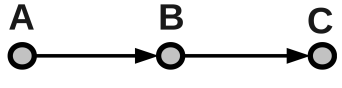


Fig. 7.18: Three Nodes Chain Scenario

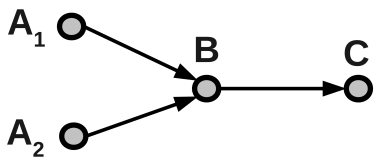


Fig. 7.19: Contending Nodes Scenario

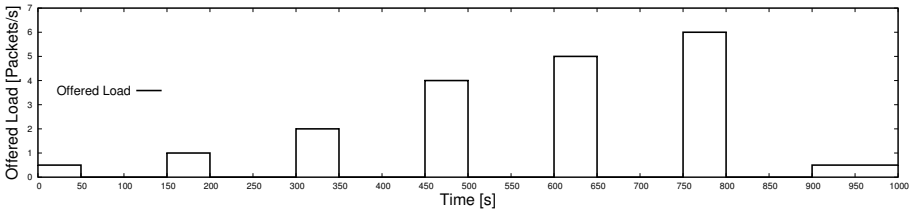


Fig. 7.20: Offered Load in the *Three Nodes Chain* Experiment

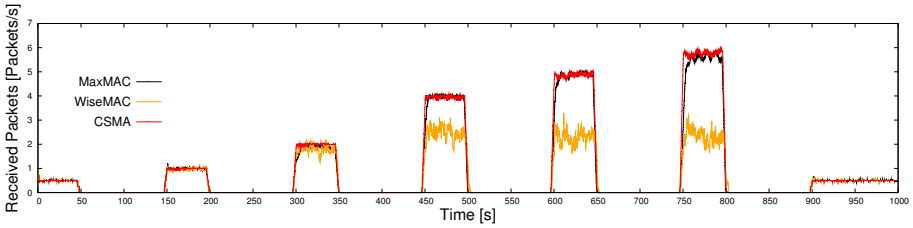


Fig. 7.21: Packet Reception Rate at Sink Node C in the *Three Nodes Chain* Experiment

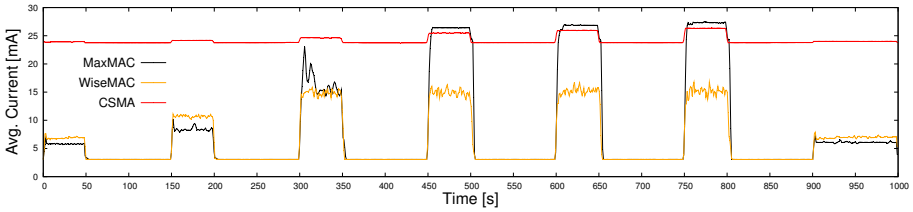


Fig. 7.22: Average Current of Node B in the *Three Nodes Chain* Experiment

routine that is called for every received byte (each 416 µs), we can safely assume that the accuracy of the synchronization remains within less than 1 ms. Having the DCOs of the nodes synchronized, we measured the delays by subtracting the time the packet is received by the sink from the packet generation time.

As one can clearly see comparing the received packets with the offered load, CSMA manages to handle almost all packets from *A* to *C*. It only suffers minor losses at the load peaks. WiseMAC's throughput stalls at a maximum of 3 packets/s, which corresponds to roughly 50% of that of CSMA. Figure 7.21 clearly shows that MaxMAC with its state-based run-time traffic adaptation mechanism reaches the same throughput as the energy-unconstrained CSMA. As the MaxMAC protocol adaptively allocates more duty-cycles and switches to CSMA at a rate of $T_{CSMA} = 3$ packets/s, it manages the load peaks without major packet loss.

Figure 7.22 depicts the mean current draw of node *B*, averaged over 20 measurements and filtered using a Central Moving Average Filter of 1s. One can clearly see the big gap in the current draw between the E^2-MAC protocols WiseMAC and MaxMAC versus the energy-unconstrained CSMA protocol. With low traffic, CSMA wastes a lot of energy on idle listening. The load peaks are hardly visible at all, as the transceiver does not consume much more power when transmitting, compared to idle listening. As MaxMAC switches to the CSMA state with the rate reaching $T_{CSMA} = 3$ packets/s, the power consumption of MaxMAC accordingly jumps to the level of CSMA, too. Thanks to the run-time traffic adaptivity mech-

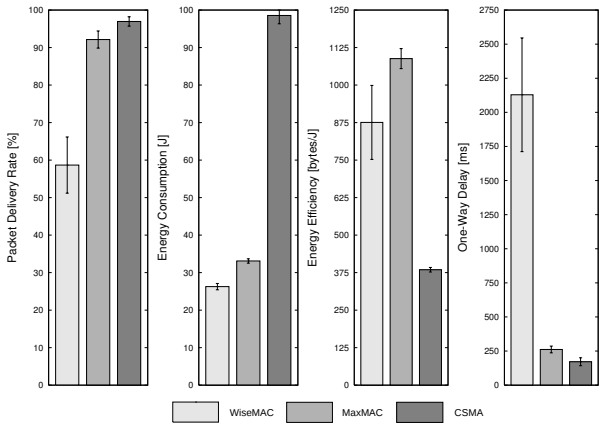


Fig. 7.23: PDR, Energy Consumption, Energy-Efficiency and One-Way Delay in the *Three Nodes Chain* Experiment

anisms, MaxMAC reaches the same energy-efficiency in the low-traffic-phases as WiseMAC, but is able to handle the load peaks with lower packet loss. The *on-demand* scheme of MaxMAC further succeeds well when the packet rate decreases, where MaxMAC quickly falls back to the Base Interval state.

Figure 7.23 depicts the packet delivery rate (PDR), the overall energy consumption of node *B*, the energy-efficiency and the one-way delay from *A* to *C* in the *Three Nodes Chain* experiment. One can easily see that MaxMAC with a PDR of more than 92.2% achieves a slightly lower PDR than energy-unconstrained CSMA (96.9%), but a far higher PDR than WiseMAC (58.7%). Most of the losses in the MaxMAC experiment runs occurred right at the beginning of the load bursts. As the load exceeds the predefined threshold values, some initial time is necessary to change from the duty-cycling states to the CSMA state, during which most packets were lost. Since the load thresholds have to be exceeded on all the nodes in the chain, congestion effects can occur in the beginning of a load burst, where the nodes close to the sink have not yet changed the state, which finally may result in buffer overflows or collisions with transmissions on the second link.

With MaxMAC allocating the radio transceiver *on demand*, and quickly falling back to the *Base Interval* state when the load decreases, the total energy consumed by node *B* amounted to just 33.1 J, as opposed to 98.5 J for CSMA. The energy-efficiency is measured as the total amount of bytes transmitted across node B per consumed Joule of node B, since only node B is measured by the SNMD device. The measured values reveal that MaxMAC achieves an even better efficiency ratio than WiseMAC. Keeping the radio on during the high-traffic phases obviously pays off with respect to the overall efficiency as well, since the higher energy costs are in such situations legitimized by the higher achievable throughput. The evaluation of the one-way delay revealed that WiseMAC sticking to its strict duty-cycling pattern with sampling the channel each T=500 ms exhibited a far higher average one-way delay (2.1s) than MaxMAC (261 ms) or CSMA (171 ms). With WiseMAC,

incoming packets are buffered in the transmit queue in every node, and transmitted in a burst as soon as the channel contention has been won, which has a deteriorating impact on the end-to-end latency. CSMA and MaxMAC (at rates $\geq$ 3 packets/s) can rely on the next node constantly being awake and hence immediately transmit any incoming packet. Thus, they do not need to wait for the next wake-up of the intermediate node and can neglect to transmit long preambles, which generally keeps the end-to-end latency low.

Contending Nodes Scenario

In a second small-scale experiment, we examined the protocol's behavior under variable traffic from two contending nodes. The nodes A_1, A_2 were configured to generate variable load, which is forwarded via node *B* to the sink node *C* (c.f Figure 7.19). All nodes were again kept on a table, and node *B*'s current was measured with an SNMD device. The shape of the offered load generated at nodes A_1,A_2 is depicted in Figure 7.24. It was chosen to illustrate the behavior of the protocols during phases where neither A_1 nor A_2 is generating load, where *one* of the source nodes is generating load, and when *both* nodes are generating load and hence have to *contend* for the medium access. Figure 7.25 depicts the rate of received packets at the sink node *C* over time, filtered with a Central Moving Average Filter of 1s and averaged across 20 experiment runs.

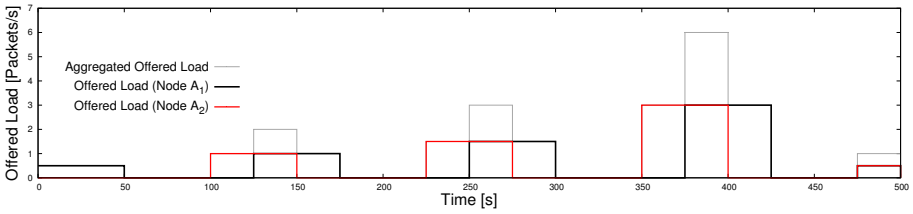


Fig. 7.24: Offered Load in the *Contending Nodes* Experiment

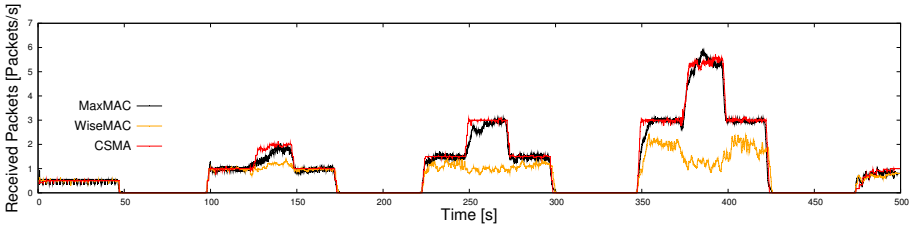


Fig. 7.25: Packet Reception Rate at Sink Node C in the *Contending Nodes* Experiment

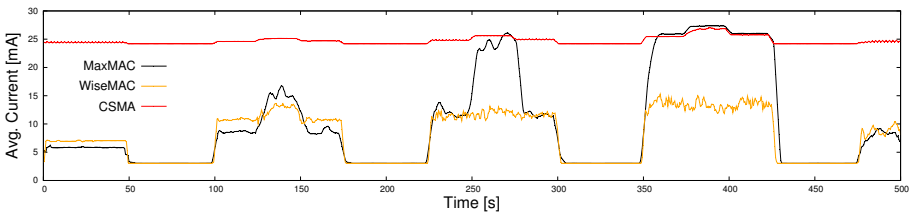


Fig. 7.26: Average Current of Node B in the *Contending Nodes* Experiment

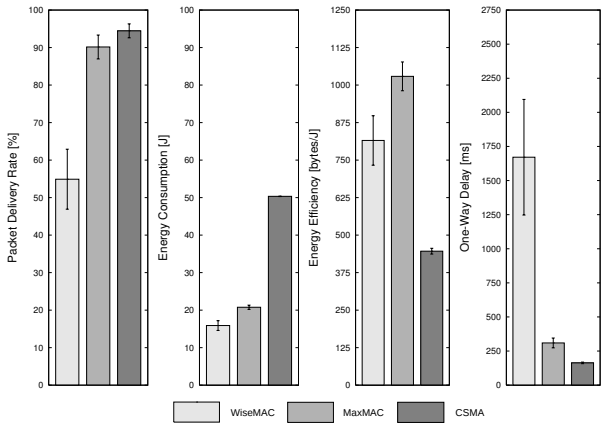


Fig. 7.27: PDR, Energy Consumption, Energy-Efficiency and One-Way Delay in the *Contending Nodes* Experiment

Figure 7.26 depicts the average current draw of node *B* (mean of 20 measurements captured with an SNMD). One can again see the large gap in the average current draw between the E^2-MAC protocols WiseMAC and MaxMAC and the energy-unconstrained CSMA protocol. Since receiving and transmitting are similarly expensive, the load peaks do not have a large impact on the CSMA curve. As Figure 7.25 clearly exhibits, WiseMAC does not exceed a rate of 1.5 packets/s. Its performance degrades in case of high contention at higher traffic rates. During the load peaks with both nodes sending at increased rate (i.e., at t=200s and t=375s), the effective throughput degrades, as transmission attempts from A_1,A_2, but also *B* forwarding received packets to *C*, increasingly cause collisions. Since in WiseMAC, transmission opportunities are limited to two wake-ups per second, collisions are more likely to occur than in the case of MaxMAC, where the additional wake-ups of node *B* alleviate its role as a throughput-restraining bottleneck.

Figure 7.27 depicts the PDR, the average energy consumption of node *B*, the energy-efficiency measured at node *B* and the one-way delay from *A* to *C*, in analogy to the results of the former experiment. The results exhibit a similar picture: MaxMAC with a PDR of roughly 90.3% achieves a slightly lower PDR than energy-unconstrained CSMA (94.5%), but a far higher PDR than WiseMAC (54.9%). Most of the losses within the MaxMAC runs proved to occur at the start of the load bursts and during the contention phases where both source nodes A_1, A_2 generated traffic. The evaluation of the energy consumption of node *B* and the one-way delays exhibited similar results as the previous experiment. MaxMAC reaches a similar PDR and latency as CSMA, however, at the energy cost of less than 50% of the latter. It also exhibits the highest energy-efficiency in terms of transmitted bytes per consumed Joule at node *B*. WiseMAC wastes a lot of energy on retransmissions caused by competing medium access. In contrast, MaxMAC delivers the major portion of packets from the two sources to the sink during the load peaks, which positively impacts on its efficiency metric.

Simulation Results vs. Tabletop Experiments

A general observation from the first two experimental scenarios is the astonishing similarity between the simulation results and the real-world results gained in the tabletop experiments. When comparing Figures 7.22 and 7.26 with Figures 7.4, 7.5, 7.6 of the simulation experiment, where a similar experiment with timely variable traffic is conducted, the similarities between the throughput and power consumption curves become obvious. Although the parameters of the simulation were different (115'200 bps in simulation vs. 19'200 bps in the real-world environment), the resulting ratios between the achieved maximum throughput of the protocols WiseMAC, MaxMAC and CSMA are in a comparable range. The behavior of MaxMAC with respect to triggering the adaptation mechanisms are well recognizable in simulation and in the real-world experiment.

7.4.4 Distributed Testbed Experiments

The major advantage of MaxMAC is its ability to switch between the primary objective of energy conservation and the throughput and latency characteristics of energy-unconstrained CSMA, depending on the load conditions. In order to examine and verify this property across more than two hops, nodes need to be physically separated, such that transmissions from one node do not impact too heavily on all other nodes, e.g., on nodes exchanging packets on the far other end of the network. In order to achieve a certain *spatial reuse* of the channel, not all nodes should hence be located within each other's transmission range.

We hence set up an indoor distributed testbed with 7 MSB430 nodes distributed across four floors of our institute building, as schematically depicted in Figure 7.28. All nodes were placed in different rooms of the building to obtain a network where nodes communicate across concrete walls and floors. Figure 7.28 depicts the links that were used throughout the evaluation, which yielded a high packet delivery rate

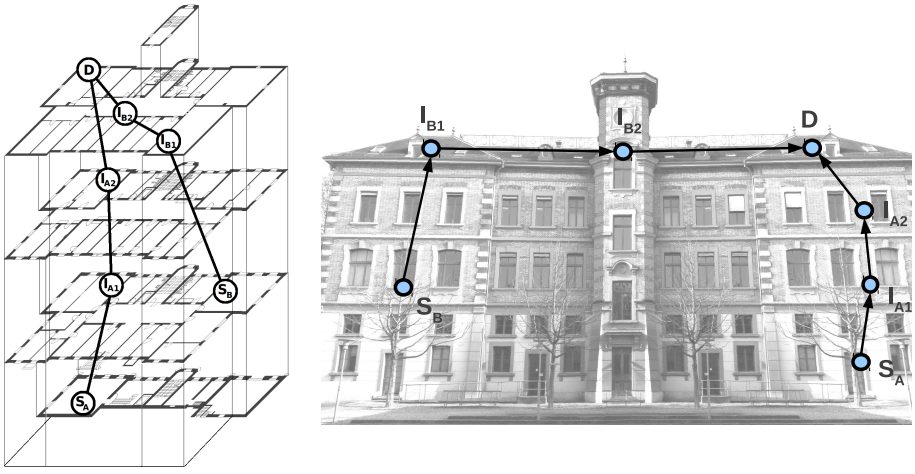


Fig. 7.28: Indoor Distributed Testbed Scenario

($\geq$95%) in case of no other ongoing traffic. It was naturally impossible to perfectly *shield* the links from each other, but a certain *spatial reuse* could nevertheless be achieved. For all the subsequent experiments and evaluations of this section, we made use of our testbed management architecture TARWIS, which is discussed in detail in Chapter 3. TARWIS allows for repeatedly running a large number of experiments without having to be physically present and without any continuous interaction with the testbed, which massively facilitated and expedited experimentation, not only for the evaluations of this chapter.

Multi-Hop Chain Scenario

In a first experiment, we evaluated the maximum achievable throughput of the three MAC protocols across a chain of 4 nodes (S_A to D) and 5 nodes (S_A to I_{B2}). Traffic was generated at different load levels at node S_A at t=90s during 60s. Figure 7.29 depicts the seven different load curves of node S_A with the examined load peaks of 0.5, 1, 2, 3, 4, 5 and 6 packets/s. Before the load peak starting at t=90s, the source node S_A sends one packet each 10s towards the destination. After the load peak, the load falls back to zero until the experiment ends at t=260s. This experiment was designed to examine the run-time behavior of the protocols when peaks of different intensities in the network load occur, and to determine maximum throughput and end-to-end latency of the three protocols across multiple hops.

Each offered load setting was examined with 20 independent runs. During each experiment run, each node estimates its energy consumption using our software-based energy estimation framework [86], which is discussed in detail in Chapter 4. Nodes print their energy consumption estimation to the serial interface every 5s, which are collected by TARWIS (cf. Chapter 3). Every node in the network was measured and calibrated individually in advance of deployment, which permitted us to apply the *per-node* calibrated parameter values determined in [86]. With

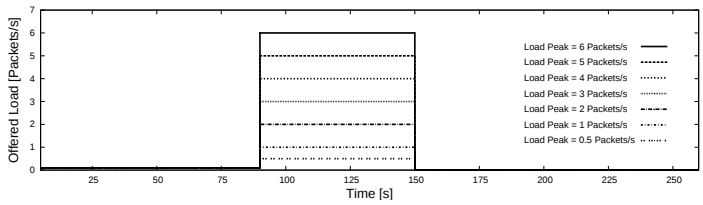


Fig. 7.29: Traffic Shapes with different Load Rates

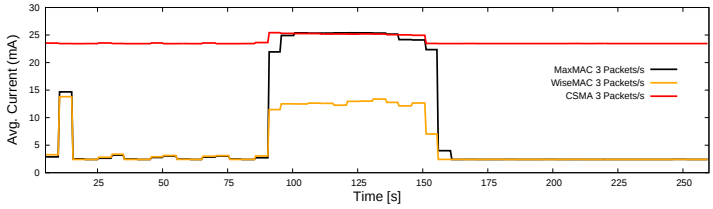


Fig. 7.30: Software-based Energy Estimation, using Load Rate of 3 Packets/s

these values, the mean absolute estimation error (MAE) across all examined traffic rates remained in the range of 1%, which we consider sufficient for evaluating and comparing our MAC protocol prototypes' performances. Figure 7.30 depicts the resulting average current draw of node I_{A1} over the experiment duration, using the results of the 4-nodes experiment runs with the peak load rate of 3 packets/s. The curves of the three protocols very much resemble the former curves from the small-scale experiments in Section 7.4.3. The step-like shape of the curve stems from the much lower resolution of only one sample obtained each 5s (=0.2 Hz), as opposed to 1000 Hz with the SNMD. The software-based methodology in general is not well-suited to examine protocol traces over time with a high resolution, but it suffices well for estimating the total energy consumption over an experiment period or an interval of several seconds. Using the software-based estimations, the impact of the load peaks generated during 60s starting at t=90s and the reaction of the protocols become clearly visible. MaxMAC is able to deliver the same load as CSMA, and falls back to the default behavior after the load peak. WiseMAC has a lower average current draw during the load peak, but only delivers a fraction of the offered load of that of CSMA or MaxMAC. With WiseMAC and MaxMAC, the initial full-preamble broadcast along the node chain is well visible in the node's energy consumptions. As nodes do not yet *know* each other's schedule offsets, the first packet traversing the chain triggers a series of broadcasts with long preambles (500ms) in all participating nodes, which are overheard by some of the nodes in the chain. This causes the significant spike in the average current at the experiment start, which is well-visible roughly at t=10s in Figure 7.30.

Figure 7.31 depicts the obtained throughput during the load phase at the sink nodes. One can clearly see that CSMA and MaxMAC both succeed in delivering almost the full offered load up to a rate of 4.5 packets/s for the 4 nodes experiment, and 3.5 packets/s for the 5 nodes experiment. Due to the fact that all nodes transmit and listen to the same channel, and imperfect spatial reuse in the indoor testbed (e.g.,

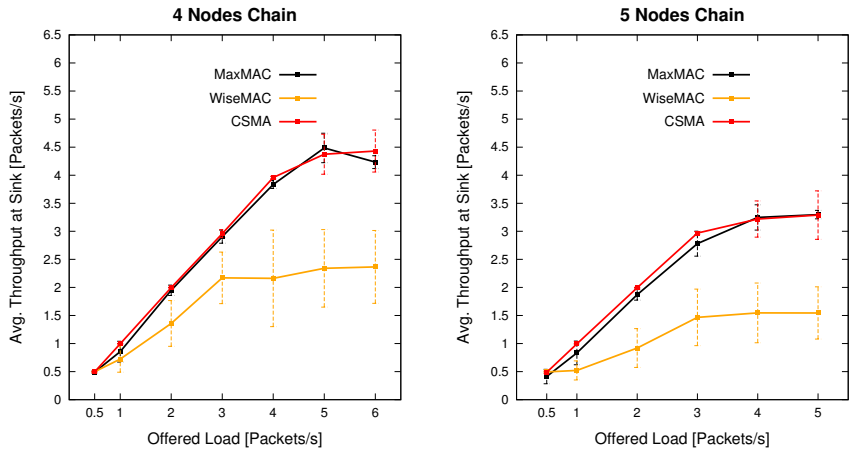


Fig. 7.31: Throughput dependent on Offered Load

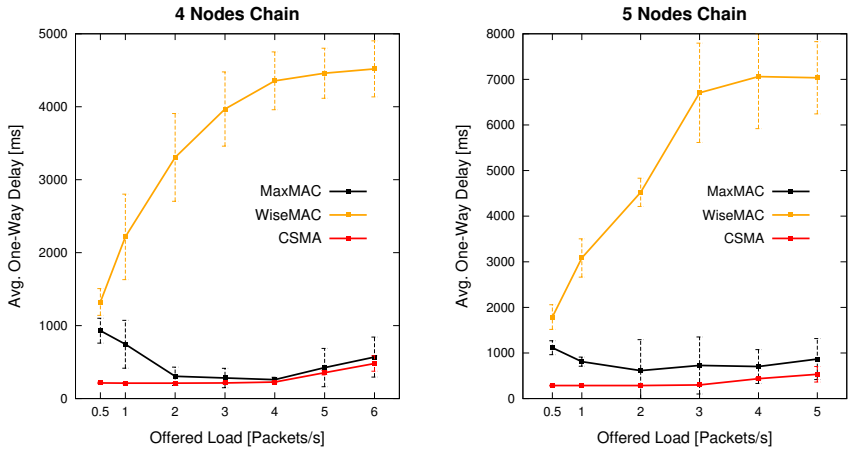


Fig. 7.32: One-Way Delay dependent on Traffic Rate

transmissions from I_{B2} *to* D impact to a non-negligible extent also on the nodes I_{A2} and I_{A1}), the maximum throughput across the 60s load peak reaches only 3.5 packets/s. Figure 7.32 depicts the latency measured from source to sink vs. traffic rate in the 4 nodes and 5 nodes scenario. MaxMAC's latency decreases for rates 0.5 to 4 packets/s. With increasing rate, the protocol allocates *Extra Wake-Ups* and switches to CSMA, which pushes the delay into the range of that of CSMA. With rates above 4 packets/s, congestion effects lead to a significant increase of the one-way delay. CSMA suffers from the same congestion effects at high rates as well, however, exhibits a much lower delay at low traffic rates. The one-way delay of WiseMAC significantly increases with the increasing traffic rate. Since WiseMAC is limited to few transmission opportunities (two wake-ups per second), the waiting time for forwarding packets is higher than in MaxMAC/CSMA. Additionally, failing contention attempts and collisions are more probable and frequent than in MaxMAC/CSMA. At higher load rates, WiseMAC tends to hopwise queue packets and transmit them in a burst, which significantly increases the end-to-end latency.

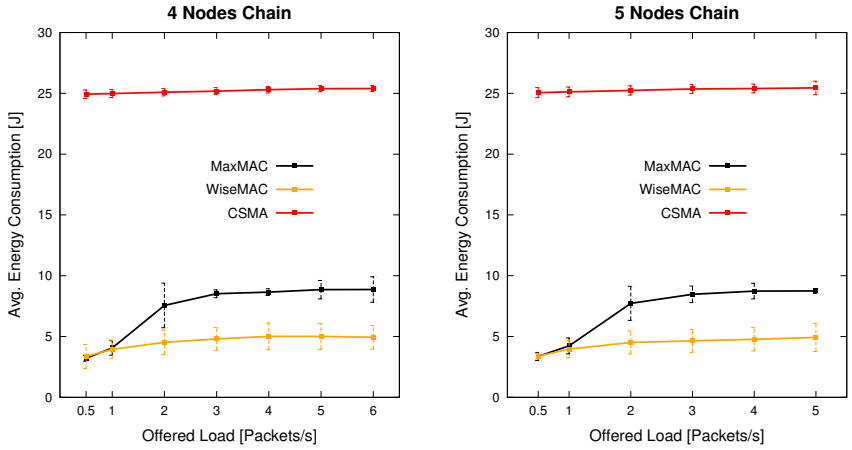


Fig. 7.33: Average per-Node Energy Consumption over the 260 Seconds

Figure 7.33 depicts the average energy consumed by each node of the chain scenario. While the energy consumption of CSMA remains constantly high at a level of 25 J, independent of the traffic load, the total energy consumption of MaxMAC increases with the load rate, but peaks at roughly 10 J. This peak is not exceeded, as MaxMAC falls back to the default *Base Interval* state after the load peak, where it only samples the channel every T=500 ms and hence saves a major portion of the energy spent in CSMA. WiseMAC consumes less energy for all load rates, however, at the cost of massively degraded maximum throughput and latency.

Variable Traffic from Leaf Nodes Scenario

In our second experiment, we evaluated the behavior of the three protocols with variable and contending traffic from different areas of the network, using the entire V-shaped network consisting in 7 MSB430 nodes depicted in Figure 7.28. The two leaf nodes S_A and S_B generate variable load across their subtrees towards the sink node D. The shape of the offered load is similar to that of A_1, A_2 in Figure 7.24 of the smaller tabletop experiment, with the minor difference that the rate during the second and third load peaks were reduced to 1.5 and 2 packets/s, cf. Figure 7.34. Due to interferences of the concurrent transmissions within the building, more generated traffic had a vastly deteriorating impact on the resulting end-to-end throughput for all protocols. We specifically chose the shape of the offered load to illustrate the behavior of the three examined protocols during phases where *neither* branch of the tree is generating load, where *one* leaf node is generating load or when *both* leaf nodes are generating load.

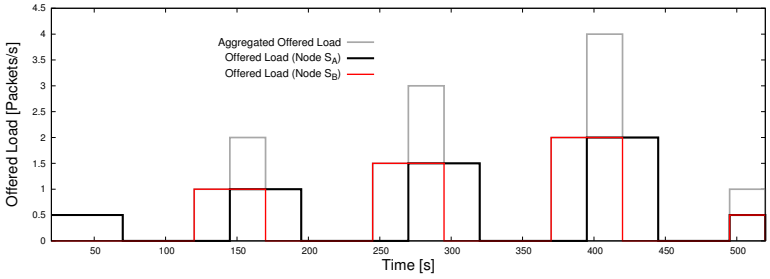


Fig. 7.34: Offered Load from Nodes S_A and S_B

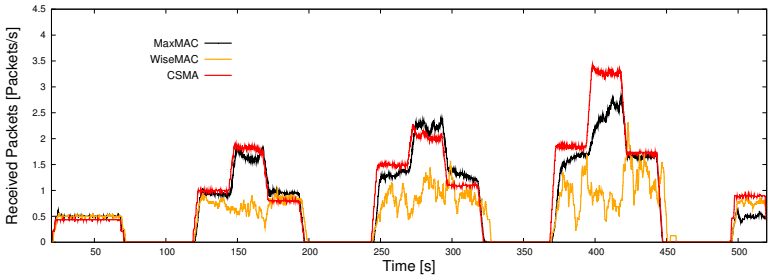


Fig. 7.35: Packet Reception Rate at Sink Node D

Figure 7.34 depicts the resulting rate of received packets by the sink node *D*. The displayed rates were again calculated using a Central Moving Average filter of 1s and computing the average of 20 independent experiment runs. One can clearly see that WiseMAC's maximum throughput stalls at slightly more than 1 packet/s, and fluctuates heavily because of the previously discovered effect of queuing and burstwise transmitting. CSMA reaches a throughput of roughly 3.5 packets/s, and MaxMAC up to 3 packets/s. Figure 7.35 clearly exhibits that MaxMAC's performance degrades with increasing contention in the distributed testbed. In larger multi-hop networks, collisions are generally more likely to occur in contention-based protocols, since a random backoff mechanism can never prevent collisions but only make them less probable. Reasons for this include, but are not limited to the transceiver switches, during which the nodes can neither receive nor transmit anything, and which can hence lead to concurrent medium access. Furthermore, the hidden node problem is not addressed at all in none of the examined protocols.

The PDR of MaxMAC in Figure 7.36 conveys a massive improvement compared to WiseMAC: applying MaxMAC's run-time traffic adaptation mechanisms seems to pay off dramatically. The PDR of MaxMAC (84.5%) in the distributed experiment involving all nodes of the testbed is significantly higher than that of WiseMAC (53%). MaxMAC's performance, however, clearly lags behind CSMA (96%), which managed to deliver the major portion of the packets across the busy network. We observed that main reason behind this performance degradation was the phenomenon that MaxMAC LEASE timeouts sometimes expired due to a series of timely-correlated collisions. When intermediate nodes in the chain fall back to the default behavior during a high-traffic phase, some time is necessary to exceed the thresholds and re-establish the fully active chain of nodes, during which most of these losses occurred. Besides the PDR, the bars of the average node's energy consumption and the energy-efficiency displayed in Figure 7.36, convey a similar improvement as in the small-scale experiments. Note that in contrast to Figures 7.23

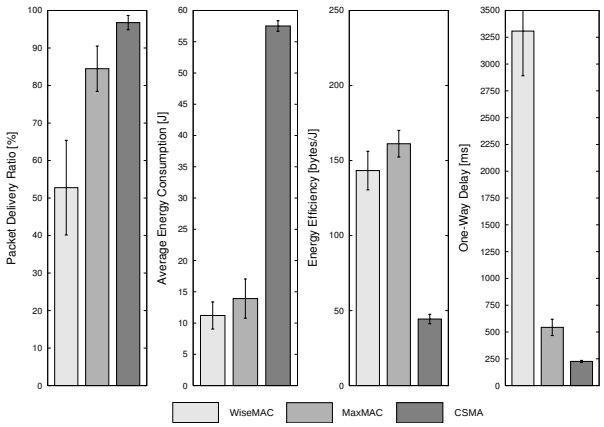


Fig. 7.36: PDR, Energy Consumption, Energy-Efficiency and One-Way Delay in the *Variable Traffic from Leaf Nodes* Experiment

and 7.27, where the efficiency is measured as the amount of bytes transmitted by node *B* per consumed Joule, Figure 7.36 depicts the transmitted bytes from *source to sink* per consumed Joule of *all nodes* in the network. Since we do not employ SNMDs in the testbed distributed environment, the consumed Joules and the efficiency in Figure 7.36 are calculated based on the software-based estimations, and hence the results of this section can not be compared to those of Section 7.4.3. The analysis of the average end-to-end latencies in the same figure indicates that WiseMAC is suffering more from congestion effects than MaxMAC or CSMA. Due to heavy channel usage and the few transmission opportunities of WiseMAC, nodes have to buffer packets for a long time until they can transmit them in a burst, which heavily impacts on the average end-to-end latency. In contrast, MaxMAC exceeding the thresholds and operating in the CSMA state can transmit queued packet trains as soon as the channel is found idle.

Figure 7.37 depicts the energy consumption estimations of the seven nodes in the network. Each group of bars depicts the seven nodes' estimations for the protocols WiseMAC, MaxMAC and CSMA, with the sink node in the middle and the subtrees to the left and to the right. The figure conveys that, across the node chains that participate in forwarding the generated load $S_B \rightarrow I_{B1} \rightarrow I_{B2} \rightarrow D$ and $S_A \rightarrow I_{A1} \rightarrow I_{A2} \rightarrow D$, the energy estimations of the nodes decrease with WiseMAC, but tend to increase with MaxMAC and CSMA. This interesting observation can be explained as follows: WiseMAC and other preamble-sampling based approaches generally shift the cost from the receivers to the senders. A sender has to *catch* the receiver in one of its short channel polls, using long preambles to compensate for clock drifts, for the contention and medium reservation mechanism and for further implementation-related inaccuracies. In WiseMAC, the further away a node is from the source node, the lower is hence its energy consumption, as the rate of received packets generally decreases with every hop because of the rather high packet loss rate. The sink then inherently exhibits the lowest energy consumption, as it does not forward any of the received packets, cf. Figure 7.37. With MaxMAC, the energy estimations tend to increase across the chain. This increase can be explained as follows: first, the closer the nodes are to the sink, the more

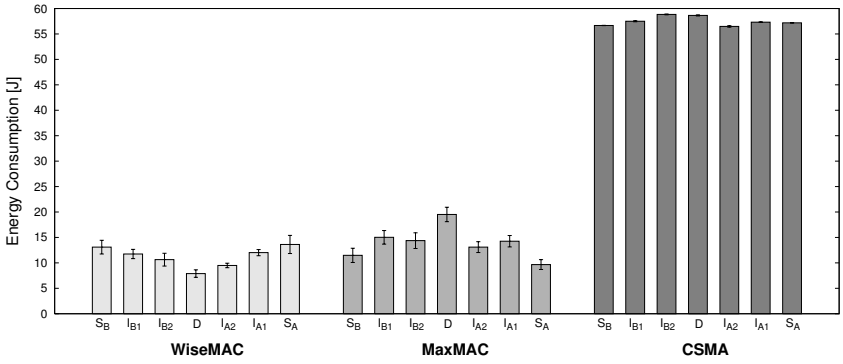


Fig. 7.37: Energy Estimations in the *Variable Traffic from Leaf Nodes* Experiment

they need to apply costly retransmission attempts due to the increasing contention in this area. Second, as the sink node receives packets from both subtrees, it exceeds the MaxMAC thresholds earlier and hence switches to energetically more expensive states (S_1, S_2, S_{CSMA}), which explains that its energy consumption is highest. With CSMA, the energy estimations exhibit a very slight increase towards the sink, which we explain with the increasing contention for the channel and overhearing in the sink area. Since transmitting is only slightly more expensive than receiving, the increase is barely measurable and may be exceeded by the measurement variation, or even the variation among the energy consumption patterns of different node instances (cf. Chapter 4).

Intruder Detection Use Case Scenario

Our third evaluation scenario is inspired by recent work on artificial-intelligence and neural-network-based intrusion detection and office monitoring systems. The proposed event detection system presented by *Wälchli et al.* [179] can, based on the data of the passive infrared sensor and the noise sensor of the employed node platform, detect the intrusion of a thief into an office. One main difficulty in the study [179] consisted in distinguishing between normal office conditions, where people are working in the office and producing background noise, and the anomalous case where the office is searched through by an intruder. The system finally managed to raise an alarm in *every* case of a simulated office intrusion with a considerably low rate of false positives. We considered this application scenario a perfect match for evaluating the MaxMAC protocol and comparing it to the two reference protocols WiseMAC and CSMA. One main requirement for the system proposed in [179] is the capability of long-term operation, probably for months or even years. Since an event is only signaled by a small alarm message, this system does does not necessarily require a high throughput, but rather a short latency and high reliability. However, with the emergence of wireless multimedia sensor networks (WMSNs) [4][7], such a system could easily be augmented by equipping nodes with small Complementary Metal Oxide Semiconductor (CMOS) cameras. WMSNs in general have gained remarkable attention in the recent past: the *cyclops* node platform proposed by *Rahimi et al.* [150] integrates a low resolution CMOS camera with the well-known MicaZ motes. Issues related to efficient compression techniques on resource-constrained embedded devices have been studied by *Lee et al.* [110]. In scenarios where nodes have to transmit image data across the network after an event is detected, e.g., in order to let the surveillance staff decide whether the detected event is a false positive or not, the provision of reasonable throughput *on demand* clearly becomes a necessity. This justifies our efforts towards Quality of Service-aware, but at the same time energy-efficient MAC protocol mechanisms.

Figure 7.38 illustrates our application-oriented scenario: the figure displays the testbed with the V-shaped network topology. The nodes in the testbed network are assumed to be part of a distributed office monitoring and intrusion detection system. Each node is assumed to be equipped with a small CMOS camera and an

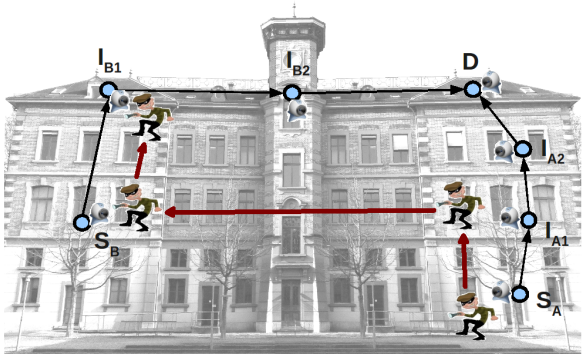


Fig. 7.38: Intruder breaking into Offices of Nodes S_A, I_{A1}, S_B, I_{B1}

infrared sensor. Based on its sensor values and prior calibration, it detects anomalous behavior as proposed in [179]. The sink node *D* is again located in the top right corner, and is assumed to be connected to the Internet to contact the facility management staff. The events we simulate are the following:

- All nodes except for the sink are generating *status* messages each 20s to inform the sink about their alive status (background traffic). In each experiment run, the initial idle period lasts for 100s.
- At t=100s after experiment start, an intruder enters the surveyed building in the ground floor and enters the office of node S_A, as displayed in the bottom right corner of Figure 7.38.
- Node S_A notices the intruder and generates an image, which is split into 100 packets (same as in the previous experiments) and sent towards the sink. The node is configured to send at a rate of 2 packets per second, hence the process takes roughly 50s.
- The intruder moves up the stairs into the first floor, where he breaks into the office of node I_{A1}, exactly 40s after visiting the first office. Node I_{A1} notices the intruder and also sends an image towards the sink
- Another 40s later, the intruder breaks into the room of node S_B located on the same floor, where the same procedure is triggered.
- Yet another 40s later, the intruder breaks into the room of node I_{B1} located in the second floor, where the same procedure is triggered. Finally, the intruder leaves the building.
- Every node, after transmitting its image data, falls back to its default behavior, generating *status* messages each 20s and sending them across the intermediate nodes towards the sink.

Figure 7.39 illustrates the shape of the offered load generated by the four sensor nodes in the different rooms of the building over experiment time. Each node starts transmitting its series of packets when the intruder is in its office. There are overlaps of duration of 10s where two nodes are concurrently attempting to send their packets towards the sink, as illustrated in the aggregated offered load curve.

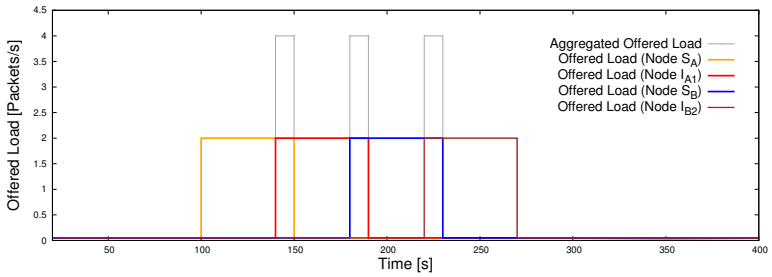


Fig. 7.39: Offered Load from Nodes S_A, I_{A1},S_B *and* I_{B1}

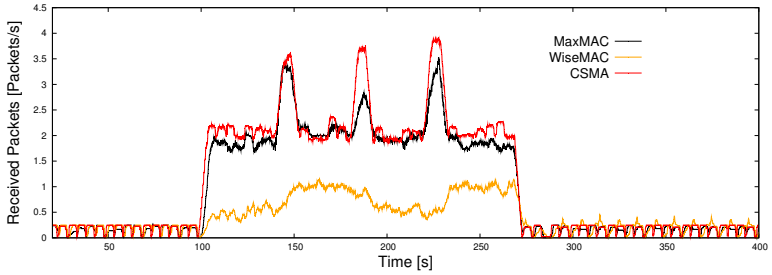


Fig. 7.40: Packet Reception Rate at Sink Node D

Since experimental evaluations of WSN mechanisms usually have a high variation, we ran the described experiment with 20 independent runs for each of the three MAC protocols and calculated the same metrics as in the previous experiments.

Figure 7.40 depicts the rate of received packets by the sink node *D*. The rates were calculated using a Central Moving Average filter of 1s and computing the average across the results of the 20 experiment runs. In general, the results of the use case oriented experiment convey similar results as the previous experiments: WiseMAC obviously manages well to deliver its periodic *alive* status messages to the sink. However, it suffers from major packet loss when the nodes have to transmit the 100 payload messages at a rate of 2 packets/s. With the wake-up interval T=500 ms, each node only wakes up twice per second. Since packets have to be forwarded across multiple hops, the rather limited channel contention mechanism and the hidden node problem lead to high packet losses. These are most likely caused by collisions and buffer overflows after failed transmission attempts. The rate of successfully delivered packets from the nodes S_A, I_{A1}, S_B, I_{B1} during the image transmission period does not exceed 1 packet/s on average, with the major share of packets being lost. After the triggered events, the periodic *alive* status packets sent every 20s are again received at the sink without major losses.

In contrast to WiseMAC, CSMA and MaxMAC succeed in delivering the periodic *alive* status messages, but also the major share of the 100 packets which are triggered by the intruder. The small time periods where two nodes are delivering their series of packets at the same time is managed best by CSMA. MaxMAC's rate of received packets reaches a slightly lower maximum throughput, and also tends to drop some packets when only one event is being handled. We have observed that

at a rate of 2 packets/s, nodes running MaxMAC tend to oscillate between the two states S_2 and CSMA. The different wake-ups patterns, as well as omitted transmissions due to failed contention attempts, can lead to nodes buffering 2 or 3 packets. When transmitting these in a burst, the CSMA threshold is exceeded and the node changes to the CSMA state. They may fall back again to S_2 after the LEASE timeout when the flow of packets has stabilized again.

Figures 7.41, 7.42 and 7.43 depict the share of packets from each originating node S_A, I_{A1},S_B,I_{B1} coming in at the sink node D. Figure 7.44 lists the most crucial metrics PDR, the average energy consumed by each node, the efficiency (measured as received bytes per consumed Joule) and the average end-to-end latency. Figure 7.41 conveys the superior performance of CSMA with respect to the achieved PDR (96%). MaxMAC is able to deliver the major portion of packets (89%), but suffers some losses during the load peaks, cf. Figure 7.42. Another interesting observation in this figure is that the overlapping load peaks do not impact on the fairness. When two nodes deliver their load towards the sink, both succeed in transmitting a similar share of their packets, despite the increased contention. In contrast, WiseMAC suffers heavily from congestion during the load peaks. The rate of 2 packets/s across a couple of hops is not manageable by the protocol, and results in buffer overflows and collisions. Nevertheless, it succeeds equally well as MaxMAC or CSMA in delivering the major portion of the periodic *alive* messages.

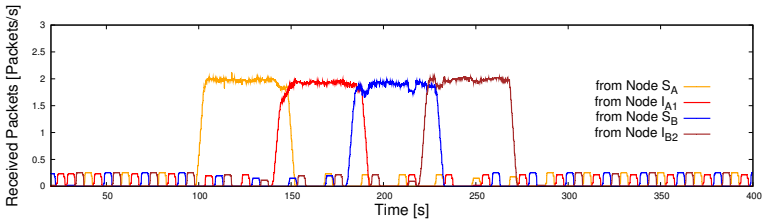


Fig. 7.41: CSMA: Packet Reception Rate from Nodes S_A, I_{A1},S_B and I_{B1}

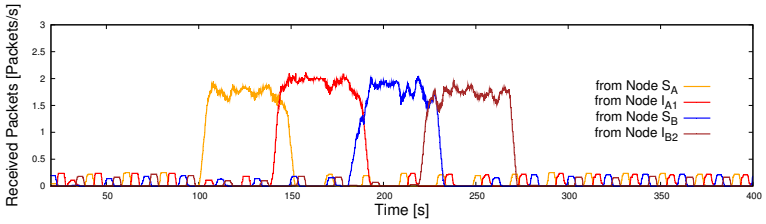


Fig. 7.42: MaxMAC: Packet Reception Rate from Nodes S_A, I_{A1},S_B and I_{B1}

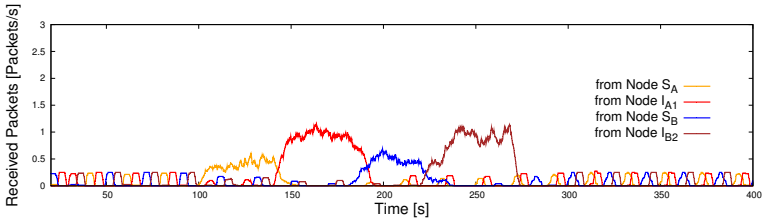


Fig. 7.43: WiseMAC: Packet Reception Rate from Nodes S_A, I_{A1},S_B and I_{B1}

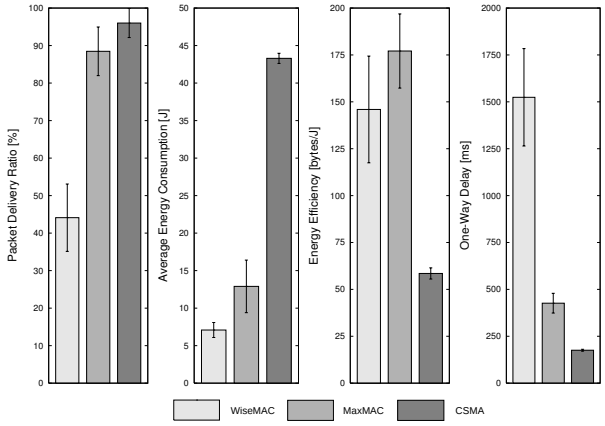


Fig. 7.44: PDR, Energy Consumption, Energy-Efficiency and One-Way Delay in the *Intruder Detection Use Case* Experiment

Figure 7.45 depicts the energy consumption estimations of the seven nodes in the network, in analogy to Figure 7.37 of the previous experiment. Each group of bars depicts the seven nodes' estimations for the protocols WiseMAC, MaxMAC and CSMA, with the sink node in the middle and the subtrees to the left and to the right. Again, the energy consumption across the node chains $S_B \rightarrow I_{B1} \rightarrow I_{B2} \rightarrow D$ and $S_A \rightarrow I_{A1} \rightarrow I_{A2} \rightarrow D$ tend to decrease with WiseMAC, but clearly increase with MaxMAC. The explanations for this behavior are basically identical to those of the previous experiment: WiseMAC shifts the transmission costs to the senders. In our scenario, I_{A1} and I_{B1} have the highest load to transmit, because they forward the packets from S_A and S_B, but at the same time also generate and forward own packets, which is different from the prior experiment. In MaxMAC, the energy costs gradually increase towards the sink. Nodes receiving packets react with *Extra Wake-Ups* and temporal switches to the CSMA state, and hence the nodes handling most traffic exhibit the highest overall energy expenditures. With CSMA, a barely measurable increase is visible but is again exceeded by the measurement variation and the variation among the different nodes' energy consumption patterns, cf. Chapter 4.

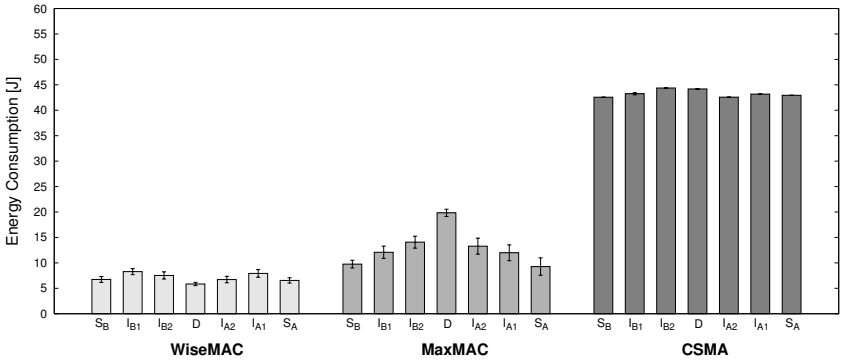


Fig. 7.45: Energy Estimations in the *Intruder Detection Use Case* Experiment

7.5 Real-world Results vs. Simulation Results

We have observed in Section 7.4.3 that there are undeniable similarities on the tabletop experiments compared to the simulation results. These similarities concern the shape of the rate of received packets over time, the average power consumption, and the relation between the different protocols' results. The maximum packet rate supported by WiseMAC evaluated to roughly 40% of that of CSMA in both simulation and the tabletop experiments, whereas MaxMAC was able to reach a similar throughput as CSMA. The same basically holds for the testbed-based results as well.

In our early case study on experiments with the first WiseMAC prototype published in [95], we have pointed out that simulation environments can be parametrized in such a way that results from small-scale real-world evaluations can be somewhat reproduced with network simulators. In contrast to the evaluation in this chapter, however, we had only conducted tabletop experiments, and have further only measured latencies and lifetimes under sparse traffic conditions, where the major shortcomings of most simulation models do not yet play a major role. These shortcomings have been pointed out for simulations in the wireless networking area in general by *Kurkowski et al.* [105] and *Andel et al.* [11], and in the sensor network MAC context in particular by *Ali et al.* [5]. They result from idealistic assumptions concerning wireless communications, such as circular transmission ranges, perfect symmetry in both directions, stable bit error rates, the availability of reliable clear-channel assessments, no processing delays due to software, and much more. For multi-hop networks, many assumptions made in network simulators do not hold anymore, which leads to a large gap between simulation results and testbed results.

When looking at the *absolute values* of the throughput and energy-consumption results obtained in the real-world testbed, we must acknowledge that the expectations raised by the simulation model could not be met. Taking into account that the bandwidth parameters of the simulations were different (115'200 bps in simulation vs. 19'200 bps in the real-world environment) by a factor of six, much higher throughput would have to be expected e.g., in the 5 nodes chain experiment. While the application of CSMA resulted in a throughput of 22 packets/s (which corresponds to roughly 3.66 packets/s when linearly taking the bandwidth reduction into account), MaxMAC and ScatterWeb^2 CSMA only reached slightly more than 3 packets/s across a chain of 5 nodes. Similarly, the efficiency coefficient of the simulation reached roughly 1800 bytes/Joule, whereas in the testbed experiment, only 10% of this value could be reached. The different factors explaining these deviations are manifold, and have been pointed out in various studies. In [95], we have shown that the scare information taken from transceiver data sheets do not realistically reflect the run-time behavior, i.e., when trying to model the transceiver switching times. Furthermore, most simulation studies only take the radio transceiver into account, a simplification which we also applied in our simulation model. However, we have observed that with the radio turned off and

the CPU in LPM1, our platform still consumes more than 2 mA, a value which almost doubles when the CPU is active and is used for computations, cf. Chapter 8, and which is definitely not negligible.

Because pinpointing the factors that cause the sometimes massive gaps between simulation results and real-world results does not form a core topic of this thesis, and since in comparison with CSMA and pure WiseMAC, the MaxMAC concept succeeded in meeting the expectations raised by the simulation results to a major extent, we neglected to further investigate the important but far-reaching research question how to bridge the gap between wireless sensor network simulation models and real-world environments.

7.6 Conclusions

In this chapter we have introduced the MaxMAC protocol, an E^2-MAC protocol aiming at high run-time traffic adaptability. We have evaluated the protocol in a networking simulator environment in different scenarios in Section 7.3. We have compared the protocol against a selection of today's most frequently cited E^2-MAC protocols under different operational traffic conditions (variable traffic with different load rates, random correlated event traffic, etc.). In Section 7.4, we have studied the real-world feasibility of the MaxMAC protocol, and compared it against two other wireless channel MAC protocols: the well-known WiseMAC protocol, and ScatterWeb^2 OS' energy-unconstrained CSMA. The evaluation relies on a series of small-scale benchmarking experiments and several experiments with different traffic patterns and networking conditions, carried out on our distributed testbed facilities operated with TARWIS, cf. Chapter 3.

Both experimental evaluations demonstrated that MaxMAC reaches its goal of being clearly distinguishable from existing preamble-sampling based approaches by reaching nearly the same throughput and a similarly low latency as energy-unconstrained CSMA, while still exhibiting the same energy-efficiency during periods of sparse network activity. The MaxMAC protocol hence succeeds in combining the advantages of energy unconstrained CSMA (high throughput, high PDR, low latency) with those of classical E^2-MAC protocols (high energy-efficiency). Like most contention-based MAC protocols, MaxMAC is a general-purpose protocol, and does not rely on assumptions which are cumbersome to achieve (e.g, rigid time-synchronization across the entire network). It can hence be applied without changes in scenarios where constant low-rate traffic is expected, and where in most cases B-MAC and X-MAC are being used today.

With B-MAC [143] and X-MAC [25] / ContikiMAC [48] being the *default* MAC layers in TinyOS and Contiki, respectively, preamble-sampling MAC protocols are nowadays by far the most widely used MAC layers in deployment studies. MaxMAC targets at improving the most significant drawbacks of such preamble-sampling MAC protocols, namely their poor performance under variable load. We

envision substantial benefits when applying MaxMAC in event-based sensor networks where at certain instants, the provision of high throughput and fast end-to-end response times becomes more important than the conservation of energy. Such applications can be found already today for instance in health-care, where nodes attached to patients need to rely on the provision of higher throughput and fast response times when critical values have been sensed, in order to communicate with central entities. The protocol furthermore facilitates real-time human interaction with sensor nodes, e.g., when querying nodes or transmitting large chunks of program code or data across several links. Chapter 9 will show that in such a case, the application of MaxMAC could yield substantial advantages.

In the following Chapter 8, we focus on a different aspect other than the energy-consumption that often plays a crucial role in WSNs. We tackle the problem of bit errors occurring as a result of degraded link qualities, e.g. because of low Signal-to-Noise (SNR) ratios, signal reflection or multipath propagation effects, or interferences with other technologies using the same or a nearby band. A key factor for the proliferation of WSN technologies is the reliability of the communication: it is crucial for many applications that the sensed data is delivered quickly and reliably across the network. Forward Error Correction (FEC) is a promising countermeasure to overcome problems related to lossy and unreliable links. However, since FEC also incur inherent and significant costs (with respect to the consumed energy and the introduced delays), their application should be limited to situations where there is a real necessity. Chapter 8 basically applies the state-based concept of stepwise allocating precious WSN resources, which was evaluated in depth in the MAC context in this chapter, to the dynamic resource allocation problem of FECs in WSNs. In this context, a run-time adaptive algorithm has to decide *when* and *where* to apply *which* kind of FEC, given an environment with timely and spatially variable link qualities.

Chapter 8

Link-Quality-Aware Adaptive Forward Error Correction Strategies

Forward Error Correction (FEC) is a mechanism for error control in data transmission over erroneous links. FEC is based on *Error Correcting Codes (ECCs)*, which can detect and correct a certain amount of bit errors in a sequence of bits, without making a retransmission necessary. FEC schemes are used today in a wide range of commercial and industrial products, where data is transmitted over erroneous channels and where, henceforth, bit errors are likely to occur. They are applied in various contexts and in many commercial consumer products: applications range from CD-ROM drives, NAND flash memory devices, mass storage devices to satellite communications and digital terrestrial broadcasting. In the WSN area, research faces several challenges, where FEC schemes could be employed as an effective countermeasure, namely the inherently unreliable wireless channel, which is prone to higher bit error rates than wireline communication channels.

The contributions of this chapter published in [94][17] are basically threefold: we study the performance of a wide range of ECCs initially preconfigured in a *statical* manner in several real-world WSN experiments. We implemented a library of eight different ECCs, ranging from simple repetition schemes over Hamming-based codes to complex and powerful Bose-Chaudhuri-Hocquenghem (BCH) codes. The fundamentals of the implemented codes are discussed in detail in Chapter 2. Second, we investigate the potential of *run-time adaptive* FEC selection strategies. We developed three adaptive FEC strategies, which adapt the correctional power of the underlying ECCs depending on changes in the link quality, according to a similar state-based concept as employed in Chapter 7. We search for a strategy that fits best to cope with the particularities of WSNs, where link qualities are expected to vary over time and across different parts of the network. Third, we make the library of ECCs *libECC* tailored for the most frequent microchip on WSN platforms, the TI MSP430 [171], publicly available on our research group's website [93], since an open source ECC library for WSN nodes has to date been missing.

All experiments were carried out on our distributed testbed facilities that are operated by TARWIS, the testbed management system discussed in detail in Chapter 3.

Section 8.1 motivates our investigations on the potential of FEC in WSNs in general, and adaptive FEC schemes in particular. Section 8.2 discusses the selected implemented ECCs within *libECC*. Section 8.3 illustrates the proposed adaptive FEC schemes, which are - together with the static ECC schemes - evaluated in different real-world experiments in Section 8.4. Section 8.5 concludes this chapter.

8.1 Motivation

WSNs are growing in popularity for various applications: they are increasingly used in healthcare, in business automation and logistics, the automotive industries or as central technology in various research projects of the natural sciences. A key factor for the proliferation of WSN technologies is the reliability of the communication: it is crucial for many applications that the sensed data is delivered quickly and reliably across the network. The low-power wireless channel used in WSNs is, however, prone to a wide range of wireless phenomena. High bit error rates are often encountered due to multipath propagation, reflection and scattering effects or interferences with nearby nodes or other devices. High error rates on the link level inevitably lead to a higher rate of corrupted packets, rendering the retrieved data unusable and making costly (probably even end-to-end) retransmissions necessary.

Automatic Repeat reQuest vs. Forward Error Correction

The simplest and most naive way to deal with transmission errors on the link layer is to retransmit the same packet again until it is received without errors or a predefined maximum retry count is reached. RFC 3366 [62] describes different *Automatic Repeat reQuest (ARQ)* schemes, that are widely used today in all kinds of networks, ranging from wireless networks to Ethernet, wireline and optical networks. In ARQ, the sender appends a cyclic redundancy checksum (CRC) [141] to the transmitted packet and waits for the acknowledgement (ACK) from the receiver. In order to reliably determine the integrity of the packet, the receiver calculates the CRC across the received payload again and compares it to the received checksum. If both CRCs match, the receiver confirms the successful reception to the sender with an ACK. If the sender does not receive an ACK within a certain time window, it assumes that the transmission attempt has failed and invokes a retransmission.

A sophisticated mechanism to cope with packet corruption due to bit errors is the concept of *Forward Error Correction (FEC)* introduced in the pioneering paper by *Shannon* in [163]. FEC is used in in a wide range of commercial and industrial products where data is transmitted over erroneous channels and where, henceforth, bit errors are likely to occur. As discussed in more detail in Section 2.5 of Chapter 2, FEC mechanisms generally consist in the sender computing parity information according to the applied ECC over the data bits, and adding this redundant information to the payload. At the receiver, the decoder of the applied ECC checks the received data bits for errors by taking this parity information into account. FEC

schemes hence generally introduce an overhead with respect to computation and data to be transmitted. However, this overhead can pay off with an increased packet delivery rate (PDR) and a reduction of the (re)transmission overhead and latency, since the correction can immediately take place after packet reception. This vital advantage is also appealing in WSNs, but has yet been studied only superficially. Our recent analysis of related work and literature on FEC mechanisms conveyed that studies in this field are rather limited (cf. Section 2.5 of Chapter 2), which motivated us to design and thoroughly evaluate several FEC strategies. *Liang et al.* [114] is the most recent study on ECCs, and appeared during our own evaluations on this topic. The study examines one Hamming and one Reed-Solomon variant to protect transmissions of a TelosB network from interferences with an IEEE 802.11b/g wireless LAN, which operates in the same 2.4 GHz ISM band. The study however does not introduce a sophisticated adaptivity concept that adapts the redundancy to the encountered error rate.

Towards Run-time Adaptive Forward Error Correction Strategies

WSNs are typically configured for their intended deployment scenario at compile-time, which can lead to suboptimal parameter settings if the discovered network conditions deviate significantly from the expectations prior to network deployment. In practice, it is rather impossible to predict the frequency and severity of signal distortions, and hence the probability of bit errors and the patterns with which they occur. Therefore, it is also impossible to choose an adequate ECC code that exhibits "just enough" correctional power in advance of network deployment. The application of powerful ECCs makes sense when the channel exhibits a high BER, but it constitutes a waste of resources in case the network's link qualities are good. By facing the challenge of selecting the *best* ECC for a given link and channel in our own indoor WSN testbed, we found that the application of run-time adaptive FEC mechanisms for WSNs operating under timely and spatially variable channel conditions has generally not been studied at all. Dynamic FEC schemes allocating the correctional power of ECCs in an *on-demand* manner could be a viable alternative to static FEC with network-wide ECC configuration, where the link conditions are not taken into account at run-time in any form.

Related work to the topic of FEC in the WSN context is discussed in more detail in Section 2.5 of Chapter 2. To the best of our knowledge, extensive real-world experiences with adaptive FEC schemes applied in sensor networks operating with short range low-power wireless channels do not exist to date. Some work on adaptive FEC schemes was conducted by *Ahn et al.* [1], however, in the context of IEEE 802.11 mobile ad hoc networks, using the network simulator ns-2 [133] and a generic wireless channel error model. A few other studies have been conducted in the WSN context, but none of them in real-world environments, cf. Section 2.5 of Chapter 2. In the recent past, however, the application of simulation tools has been identified as a general drawback of many ad hoc and sensor network studies. The trend in the research field of WSNs has matured and has clearly shifted towards ex-

perimental feasibility studies of proposed mechanisms and protocols on real-world devices. We hence analyzed our contributions consisting of eight different ECCs and several adaptive FEC strategies in various scenarios in a WSN testbed with realistic deployment topologies.

8.2 Forward Error Correction Library libECC

Our library of ECCs *libECC* [93] implemented during this study consists of eight different ECCs from four basic classes of codes, each with different degrees of redundancy and different block sizes. Figure 8.1 illustrates the block size, the correctional power per block and the amount of redundancy of these ECCs. The parameters in brackets define the particular ECC configurations, but their semantics may differ among the classes (e.g., payload/parity bits, repetition factor):

- The Repetition Code [127] is the most simple and naive method to introduce an error correction capability, since it simply stretches the number of input bits and then decodes using majority logic decoding. We refer to our implementation of the repetition code as REP(3,8) throughout this chapter, because it stretches the input bits by a factor of 3 and operates on a block unit of one byte. The advantage of REP(3,8) is the low computational overhead, which comes at the cost of the resulting large portion of parity information, in particular when comparing with more complex codes.
- The Hamming code is a linear error-correcting code invented by Richard Hamming in 1950 [74]. This simple code can be seen as a cornerstone for the development of modern ECCs. *libECC* contains the popular Hamming(7,4) code, which encodes 4 data bits into 7 encoded bits, which equals its block size.
- The Double Error Correction Triple Error Detection (DECTED) proposed by *Gulliver et al.*[69] is similar to Hamming. DECTED is able to correct up to two bit errors and can detect up to three adjacent bit errors per block. The DECTED(16,8) code implemented in *libECC* takes 8 bits as input and creates an encoded word of 16 bits, hence exhibiting comfortable block size units of 1 byte for the raw data and 2 bytes for the encoded data.
- The Bose-Chaudhuri-Hocquenghem (BCH) codes, invented in 1959 by *Hocquenghem* [79] and independently in 1960 by *Bose and Chaudhuri* [22], are the

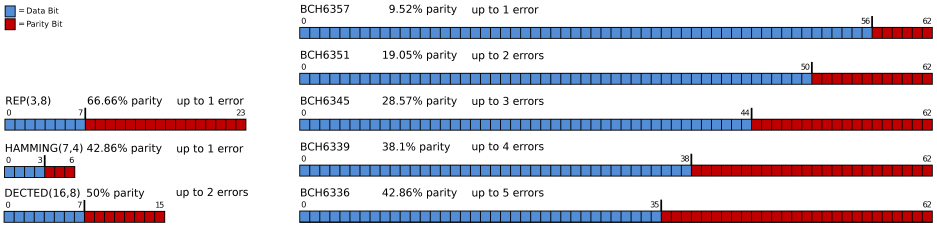


Fig. 8.1: libECC Codes: Block Size, Correctional Power, Redundancy

most complex ECCs within *libECC*. BCH codes belong to the class of *cyclic* block codes. BCH(n, k) encodes k data bits into n encoded bits. *libECC* implements BCH codes with 63 bits block size, allocating 64 bits for practical reasons, since 64 bit blocks nicely fit into the `long long` datatype. BCH(63,57), BCH(63,51), BCH(63,45), BCH(63,39), and BCH(63,36) implemented in *libECC* can correct 1,2,3,4 or 5 errors per block.

Memory Footprint

WSN nodes are heavily restrained with respect to the available computational power and memory resources. Having this restriction in mind, we cautiously kept the memory footprint of *libECC* as low as possible. Although many of the implemented ECCs require large matrices, syndrome value lookup tables and polynomials, the current version of *libECC* consumes only roughly 10 KBytes for the text segment and 326 bytes for the data segment. *libECC* allocates one statically allocated data structure, where detailed information about the decoding and correction procedures is stored after each encoding and decoding operation, e.g., the number of corrected errors, the size of the decoded payload, the duration of the decoding. *libECC* uses the basic data types of the mspgcc compiler [129], but could easily be ported to other platforms, e.g., 32-bit AMD Geode CPUs, which are frequently used in wireless mesh networks (WMNs) [139]. We have successfully tested the library on different MSP430-based WSN platforms and operating systems, such as the MSB430 [14] using ScatterWeb^2 OS [157] and TelosB [144] using Contiki OS [47].

8.3 Adaptive Forward Error Correction

WSN nodes are typically preconfigured with the most crucial parameter settings at compile-time, hence much before the actual network deployment. As pointed out in Section 8.1, this can result in suboptimal performance in case the actually encountered environment differs much from the conditions expected during planning. When deciding to apply FEC, a crucial design question consists in selecting the appropriate ECC. While a too weak code might not be able to correct many errors, a too strong code would waste precious time and energy for encoding/decoding and transmitting the additional parity data. With energy and processing power being limited resources in WSNs, this decision is of particularly high importance. The channel quality is almost certain to exhibit variations over time and can differ heavily across the different links, which renders the choice of a network-wide "optimal" ECC prior to network deployment impossible.

With our proposed adaptive FEC approaches, we address this crucial design problem of choosing an appropriate ECC code by adaptively selecting the different ECC codes for each individual link at run-time instead of applying network-wide settings prior to network deployment. In each of the adaptive FEC schemes we in-

troduce in the following, the sender decides which ECC to use based on past transmissions to the particular target node, since the channel quality and transmission success probability is usually tied to a particular link. Optimally, the selected ECC adds as little overhead as possible, but provides just enough correctional power to overcome the encountered error patterns. Figure 8.2 illustrates the finite-state-machine based concept of selecting different ECCs at run-time. The states are kept in a table for each neighbor and denote the current ECC that is used on the link to that neighbor. The ECCs contained in this chain of states are increasing in their correctional power from the leftmost to the rightmost state, and similarly with respect to computational and parity overhead. In the default *OFF* state, the node does not encode the payload at all. Hamming(7,4), can correct one error per block, DECTED(16,8) up to two errors per block, and BCH(63,45), BCH(63,39), and BCH(63,36) can correct up to 3,4, and 5 errors per block, respectively. The ability to correct multiple adjacent errors per block is a crucial advantage of the BCH variants, because random bit errors tend to occur temporally correlated, e.g., during the transmission of the same byte.

Stateful Adaptive FEC (SA-FEC)

SA-FEC is the most simple adaptive FEC mechanism one can basically think of. It selects the ECC for the next transmission to the specified destination according to the success of the last one. This means that if the last transmission of an ECC packet has been successful, SA-FEC selects the next less powerful ECC. If not, SA-FEC selects the next more powerful ECC, cf. Figure 8.2. The decision depends only on the success of the last transmission, i.e., whether a subsequent ACK has been received or not. Hence, SA-FEC only takes the very recent past into account, and reacts immediately to packet losses. The mechanism is illustrated in Figure 8.3.

Stateful History Adaptive FEC (SHA-FEC)

SHA-FEC is similar to SA-FEC, but maintains a variable denoting the currently used ECC *and* a history of entries representing the recent past transmissions, which are manipulated in a FIFO-manner, cf. Figure 8.3. In case a transmission succeeds and an ACK is received, SHA-FEC stores an integer value representing the next *lower* ECC into the history, since it assumes that a lower ECC would have sufficed as well. In case the transmission fails and no ACK is received, SHA-FEC stores a value representing the next *higher* ECC, assuming that more correctional power is

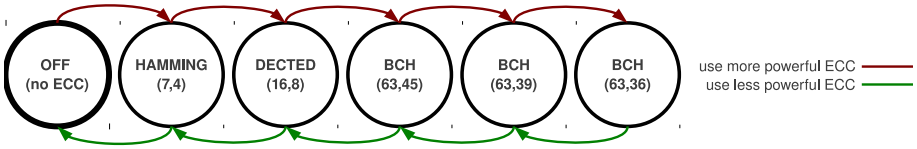


Fig. 8.2: Adaptive Forward Error Correction: ECC Selection Sequence

necessary. We sticked to a history size of five entries throughout all our evaluations. The selection of the ECC used for the next transmission is based on calculating the rounded average of the values stored in the history. As soon as the majority of entries represents the *lower* ECC, the node switches back one step in the state chain depicted in Figure 8.2. SHA-FEC reacts less quickly to link quality changes than SA-FEC, but avoids oscillation effects, i.e., switching to different ECCs after each single transmission. The history-based selection mechanism provides a means to cope with longer-term interferences, since the mechanism does not immediately fall back to a less powerful ECC after one successful transmission, but waits until a couple of transmission have succeeded and then only stepwise shifts back in the state chain illustrated in Figure 8.2.

Stateful Sender Receiver Adaptive FEC (SSRA-FEC)

SSRA-FEC extends SHA-FEC by taking into account an additional history containing *receiver* information into the ECC selection process. The entries in this history denote the maximum number of corrected errors per block e_{max} of each of the previous successful frame transmissions. Given that a packet payload consists of n blocks $b_1, b_2, \ldots b_n$, the receiver calculates for each block the number of occurred and corrected errors $e_1, e_2, \ldots e_n$ during the decoding process. In case the all the blocks could be correctly decoded, the receiver sends $e_{max} = max(e_1, e_2, \ldots e_n)$

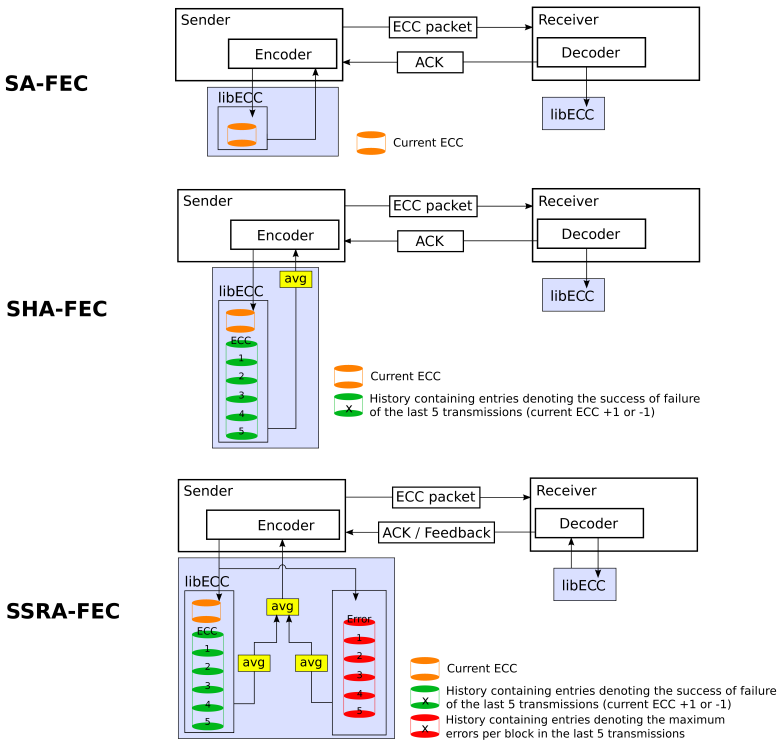


Fig. 8.3: Overview of the Adaptive FEC Concepts SA-FEC, SHA-FEC, SSRA-FEC

back to the sender in the ACK. If the frame could not be correctly decoded (which can be verified using a CRC checksum calculated over the data payload), the number of errors in the corrupted frame can obviously not be reliably determined. In order to signal packet corruption, the receiver hence neglects to send back an ACK. Just as in SHA-FEC, the sender then stores a value representing the next *higher* ECC into the history, because more correctional power is obviously necessary to cope with the current error rate in the channel.

To determine the ECC for the next transmission, the sender computes the rounded average of the unrounded averages of both histories. Basically, SSRA-FEC is an attempt to implement a *closed-loop* parameter adaptation, where not only the *sender knowledge* is taken into account, but also the *feedback* from the receiver. The mechanism involving two histories was designed to find a "suitable" ECC quickly by integrating receiver information, yet retaining a certain robustness against oscillation effects. Figure 8.3 illustrates and puts into relation the three designed adaptive FEC selection strategies we have just discussed, and which are examined in a series of experiments in the subsequent section.

8.4 Experimental Evaluation

We evaluated all implemented ECC codes of *libECC* in a statically configured manner and compared them with the three proposed adaptive FEC mechanisms SA-FEC, SHA-FEC and SSRA-FEC on the MSB430 [14] sensor node platform on top of the ScatterWeb^2 Operating System [157]. The MSB430 platform has a CC1020 [173] byte-level radio transceiver operating in the 804-940 MHz ISM frequency band. While the maximum raw bit rate of the CC1020 is 153.6 kbit/s, the utilized ScatterWeb^2 OS only supports a data rate of 19.2 kbit/s using simple on-off keying (OOK) as modulation scheme. On the MAC layer, we employed the default IEEE 802.11-like ScatterWeb^2 OS CSMA variant, which does not duty-cycle the radio in any form, cf. Section 2.1.4 of Chapter 2. We explicitly chose a non duty-cycled MAC in order to safely exclude the potential influence of radio duty-cycling on the resulting PDRs of the different ECCs examined in this section.

8.4.1 Computational Complexity

We first evaluated the ECC implementations of *libECC* with respect to computational complexity. We therefore measured the time needed for encoding different payload sizes, ranging from 3 to 52 bytes of data and for decoding the corresponding code words. Figure 8.4 depicts the encoding and decoding times for the given payload sizes. The figures display the mean values and standard deviations of 1000 encoding/decoding operations for each code in *libECC*. Obviously, the encoding and decoding durations grow linearly with the amount of payload bytes for each ECC examined. The figures clearly display a trade-off between the correctional power of the ECCs and the required computational complexity. The repetition

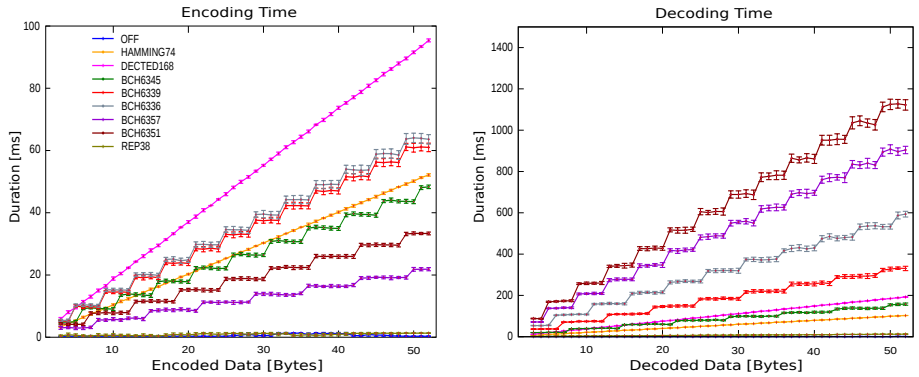


Fig. 8.4: Encoding and Decoding Time vs. Payload Bytes

code REP(3,8) for example is by far the fasted ECC, since it requires few computation and mainly consists of memory copying operations. The hidden costs of the repetition code, however, lie in the resulting large size of the parity data, which can significantly impact on the energy consumption when transmitting the frame. The BCH code exhibits a characteristic step-shaped pattern in the encoding and decoding times, which can be explained by the blockwise encoding and decoding process of the BCH code and the comparatively large block size units. The complex BCH variants with higher correctional power that add more parity information clearly exhibit longer encoding/decoding times.

8.4.2 Energy Cost Estimation

In an attempt to quantify the energy cost of the application of ECC codes, we measured the current draw of the MSB430 sensor nodes with the CPU in the two operation modes of the MSP430 [171] chip used by the ScatterWeb^2 OS, namely the fully active mode and the Low Power Mode 1 (LPM1). In LPM1, the CPU and master clock (MCLK) are disabled, but timers and peripheral interrupts are still enabled. We measured the node's current draw to account to 3.75 mA for the fully active operation mode and 1.92 mA for LPM1. Figure 8.5 depicts the measured traces and their different levels. This difference might vary by a few percent dependent on the node chosen for the measurement, since different nodes can exhibit small variations, cf. our study [86] discussed in Chapter 4. With the measured current draws, we derive a cost function to quantify the energy costs of the implemented ECCs, taking into account the time it takes to encode or decode a payload. We define $P_{Default}(t)$ as the power consumption function of the node without using FEC, and $P_{FEC}(t)$ as the respective function of the node applying FEC. We define the *cost* of the application of FEC as the *additional power* consumed while encoding and decoding. The power cost function $P_{cost}(t)$ can then be denoted as

$$P_{cost}(t) = P_{FEC}(t) - P_{Default}(t)$$

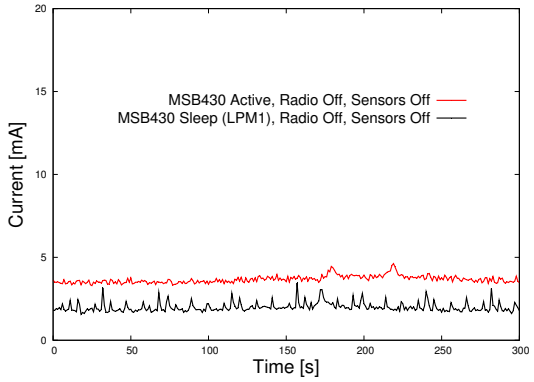


Fig. 8.5: Current Draw with fully active CPU or LPM1

Integrating the measured values of the MSB430 into the equation yields:

$$P_{cost}(t) = (I_{FEC} - I_{Default}) \cdot U_{supply} = 1.83\ mA \cdot 4\ V = 7.3\ mW$$

I_{FEC} corresponds to the average current used with the CPU in the active mode, while $I_{Default}$ is the average current used in LPM1. The encoding and decoding durations depicted in Figure 8.4 can hence be linearly mapped using the cost function $P_{cost}(t)$ to corresponding energy cost functions. For example, the energy cost $E_{enc/dec}$ of an encoding and decoding operation at sender and receiver of, e.g., a BCH(63,45) encoded payload of 32 bytes, which takes roughly T_{enc} =30 ms for encoding and T_{dec} =100 ms for decoding, calculates as

$$E_{enc/dec} = E_{enc} + E_{dec} = T_{enc} \cdot P_{cost} + T_{dec} \cdot P_{cost} = 0.95\ mJ$$

However, we also have to take into account the parity overhead to be transmitted when applying ECC. We assume that a retransmission consists in a 50 ms frame T_f and a 20 ms ACK transmission T_{ack}. With BCH(63,45) encoding, the payload is increased by 28%. However, since the frame header, preamble and postamble overhead remains constant, the entire frame size usually does not increase by more than 15%, given that the payload is in the range of 30-50 bytes. We hence assume that $T_{oh} = T_f \cdot 15\%$. Based on values measured in [86] of the mean currents of a MSB430 with the radio receiving (I_{rcv} =23.53 mA) and transmitting (I_{tx} =37.48 mA) the cost E_{oh} of the data overhead transmission and reception amount to:

$$E_{oh} = E_{oh_{snd}} + E_{oh_{rcv}} = T_{oh} \cdot (I_{tx} + I_{rx}) \cdot U_{supply} = 1.83\ mJ$$

The subsequent evaluations of this study exclusively focus on reliability measures (link-level PDRs and end-to-end PDRs) and leave aside the energy-efficiency aspect. Nevertheless, we briefly illustrate the potential *benefit* of applying ECCs with respect to the energy consumption, continuing the exemplary calculation of $E_{enc/dec}$ noted above: we again assume T_f = 50 ms and T_{ack} = 20 ms. The energy costs E_{re} of a retransmission consisting of the costs E_{snd} at the sender and E_{rcv} at the receiver account to:

$$E_{re} = E_{snd} + E_{rcv} = (T_f \cdot (I_{tx} + I_{rx}) + T_{ack} \cdot (I_{tx} + I_{rx})) \cdot U_{supply} = 17.08\ mJ$$

This cost E_{re} of an *entire retransmission* is hence more than six times higher than the cost of applying ECC, consisting in the encoding and decoding operations $E_{enc/dec}$ and the energy overhead E_{oh} for transmitting the parity data. In the presence of unreliable links with bit errors corrupting a significant share of the packets, the application of FEC may hence even make sense from an energy point of view alone. A general judgment on this question can, however, not be answered without an assumption about the channel quality and hence the frequency of bit errors, and would generally require substantial further investigations.

8.4.3 Single-Link Scenario - Indoor and Outdoor Links

This section evaluates the different ECCs of *libECC* in the two single-link scenarios, which are displayed in Figure 8.6 as links $A \rightarrow B$ and $A \rightarrow C$. Nodes A and B are placed in the two most distant offices on the same floor of the Institute of Computer Science and Applied Mathematics' main building. Node A is placed on the windowsill in one office, and the signal from its antenna has to pass five concrete walls in order to reach node B, with a distance of roughly 25m. The second link between A and C is an outdoor link with a line of sight, since both nodes are placed on the windowsills of office buildings facing each other. The line of sight distance between these two nodes is 48m.

In the single-link experiments, we study and compare the real-world performance of the different ECCs under comparable conditions without any interference from other ongoing transmissions. All measurements were captured during the night and on the weekends, in order to ensure repeatability and comparability of the experiments with the different parameter settings under equal circumstances. The use of GSM cellphones, IEEE 802.11 devices, wireless headphones, microwave ovens, or other potential sources of interference, might affect the channel quality and hence call the comparability of the results into question. Running the experiments overnight and on weekends minimizes the probability of irregular interferences caused by people working in the building, which could deteriorate the results

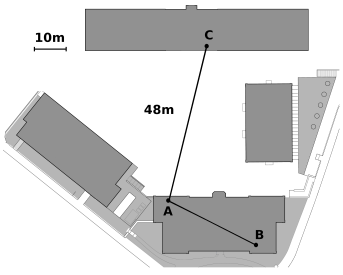


Fig. 8.6: Indoor and Outdoor Link

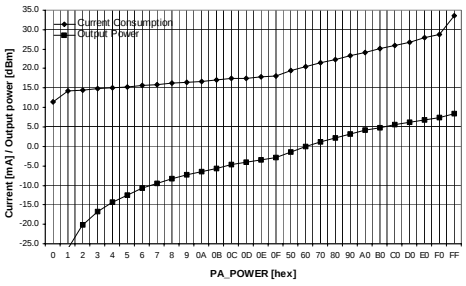


Fig. 8.7: CC1020 [173] Output Power

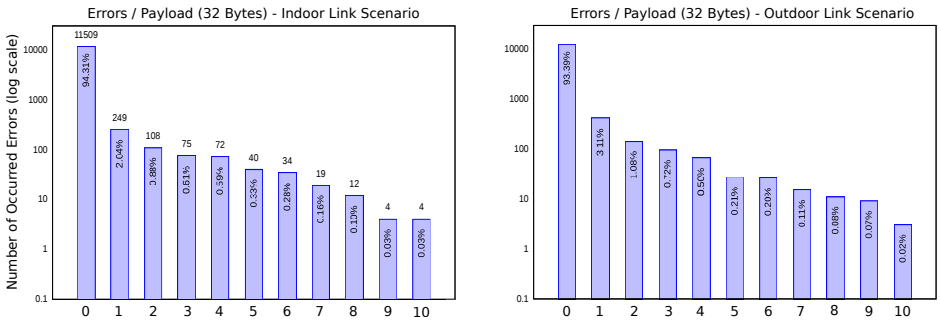


Fig. 8.8: Error Occurrence Patterns - Indoor and Outdoor Link

of one particular parameter setting, and hence call the comparability of the individual FEC approaches' results into question. Since fog or increased humidity absorb radio signals, the measurements of the outdoor link $A \rightarrow C$ were gained under dry weather conditions. We chose 1000 packets of 32 bytes (+2 bytes CRC) as the base configuration. Each three seconds, the sending node generates and encodes one packet and unicasts it to the receiver node B or C. We evaluated 15 different transmission power settings of the CC1020 chip, ranging from 1 tick ($\approx$ -25 dBm) to 15 ticks ($\approx$ -3.5 dBm), since we are interested in particular in the performance of the ECCs under different signal strengths. Figure 8.7 depicts the resulting output power of the CC1020 [173] with the different settings of the PA_POWER register.

An issue that is closely linked to the resulting ECC performance is the question what error patterns actually occur, and whether there are substantial differences in the error patterns discovered on the indoor and outdoor link. We therefore counted the number of errors per packet and examined the probability distribution of the number of errors occurring across the 32 bytes payload. We achieved this by sending an unencoded payload consisting of a predefined random bit sequence of 32 bytes that is known to the sender and the receiver and bitwise checking for errors after reception. Figure 8.8 depicts the resulting histogram of the probability distribution of 0 to 10 errors per payload, calculated across all measured transmission power settings and displayed in logarithmic scale. As one can expect, most packets did not exhibit any errors (94.31% and 93.39% on the indoor and outdoor link). The most frequently encountered error patterns were 1-bit errors (2.04% and 3.11%) - i.e., 1 error across the entire 32 bytes payload. 2-bit errors were less than half as frequent (0.88% and 1.08%), and the probability of even more bit errors occurring across the 32-bytes payload was gradually decreasing on both links, which is well observable in Figure 8.8.

Figure 8.9 depicts the PDR for each examined ECC vs. the transmission power setting for the indoor and the outdoor link. When comparing against unencoded transmission (OFF), the PDR could be increased with every examined ECC. The impact of applying FEC, however, depends heavily on the transmission power: it has significant advantages in case of lossy links with low signal strengths where errors are more frequent to occur, but does not constitute a benefit in case the signal

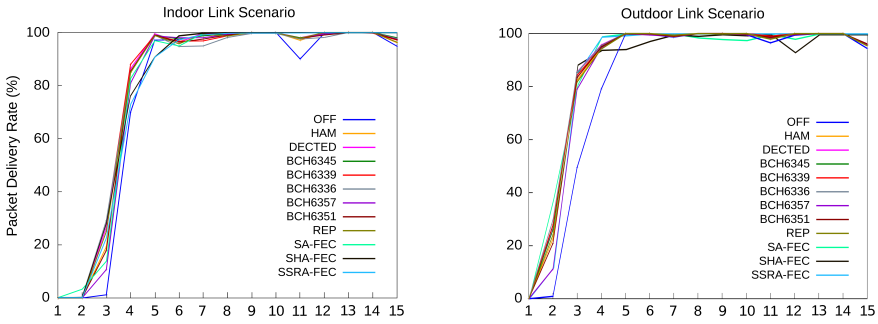


Fig. 8.9: Packet Delivery Rate vs. TX Power - Indoor and Outdoor Link

strength is strong enough to cope with minor interferences. We further observed that the most powerful ECCs did not necessarily result in a higher PDR. The most powerful codes within *libECC* did not significantly increase the PDR compared to the most simple ECCs. This indicates that the most frequently corrected errors were again 1-bit or probably 2-bit errors - an observation that was also made by *Jeong et al.* [99] and *Busse et al.* [27] using a similar radio and the same modulation scheme in comparable scenarios. A general observation is that the PDRs in the outdoor link with a line of sight connection are higher than those of the indoor link. This observation can be explained by the fact that the signal does not need to penetrate concrete walls and does not suffer from multipath propagation and reflection effects as in the indoor scenario.

Figure 8.10 depicts the PDR of the different ECCs of the outdoor link with the transmission power set to 3 ticks (≈ -17dBm) in more detail, in order to allow for a more fine-grained analysis. Again, the more powerful ECCs did not necessarily result in a higher PDR - the increment in PDR of applying FEC compared to transmitting unencoded packets is significant. Figures 8.9 and 8.10 clearly convey that the adaptive FEC mechanisms designed in Section 8.3 performed astonishingly well, since for every transmission power setting, the three proposed mechanisms achieved comparable PDR values as the static ECCs - although they only apply FEC in an *on demand* manner.

Figure 8.11 depicts the share of packets that are successfully received for each ECC scheme in the three adaptive approaches. The bars hence depict the results

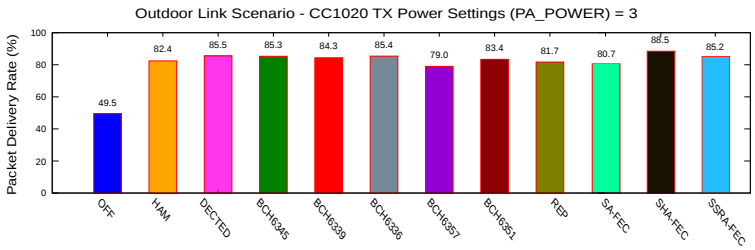


Fig. 8.10: Outdoor Link: PDRs per ECC with TX Power = 3 (≈ -17dBm)

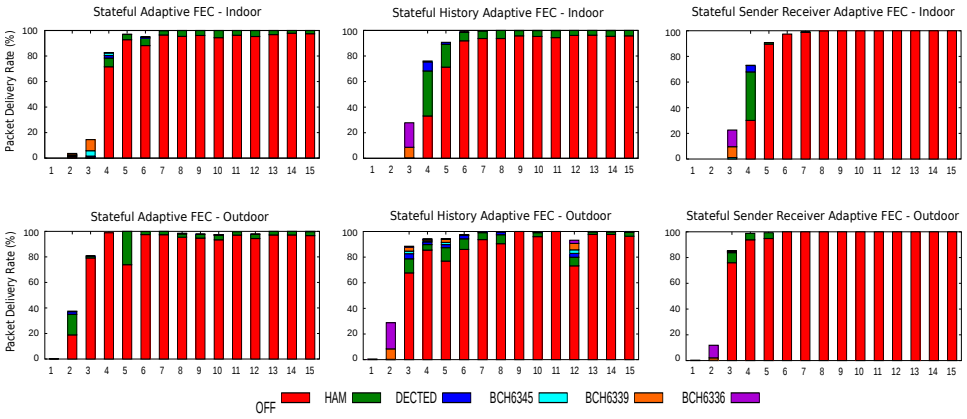


Fig. 8.11: ECC Selection of the Adaptive FEC Mechanisms: PDRs vs. TX Power

of only the three adaptive approaches of Figure 8.9 with the indoor link in the top graphs and the outdoor link on the bottom graphs. One can clearly see that the selection pattern of the three adaptive approaches are similar on both examined links, but that the three approaches differ among each other in the share of the applied different ECCs. SA-FEC reacts to packet loss quite quickly by changing to more powerful ECCs, which explains its larger share of Hamming(7,4) across all transmission power settings. SA-FEC changes to a stronger code as soon as it does not receive an ACK for the last packet, whereas in SHA-FEC and SSRA-FEC, one or more histories of items representing the transmission failures with the employed ECC has to be filled, before the code is finally changed. Since in SSRA-FEC, the decision to change to a more powerful ECC also depends on the reception of a subsequent acknowledgement indicating the amount of errors, the SSRA-FEC does not switch the code as quickly as SA-FEC and SHA-FEC, which makes its ECC selection generally more robust against oscillation.

In general, Figure 8.11 conveys that the targeted behavior of the adaptive approaches has been accomplished: the proposed run-time adaptive ECC selection schemes apply FEC more when the link is lossy (low power settings $\leq$ 6 ticks) and less when the link is strong (high power settings $\geq$ 7 ticks). For transmission power values above 7 ticks, the vast majority of packets is sent unencoded. The application of ECC has been limited to short time intervals where, probably due to random short-lived interference, packet errors were more probable to occur. Few energy and time is generally wasted for unnecessary encoding and decoding.

8.4.4 Distributed Multi-Hop Scenario

In order to examine the different ECCs and the adaptive FEC approaches in an environment that comes close to real-world conditions, we evaluated each static ECC setting and each adaptive FEC scheme in our distributed testbed of seven MSB430 nodes in a multi-hop topology. We used our fully automated testbed and experiment management system TARWIS [92] for reprogramming the sensor nodes and

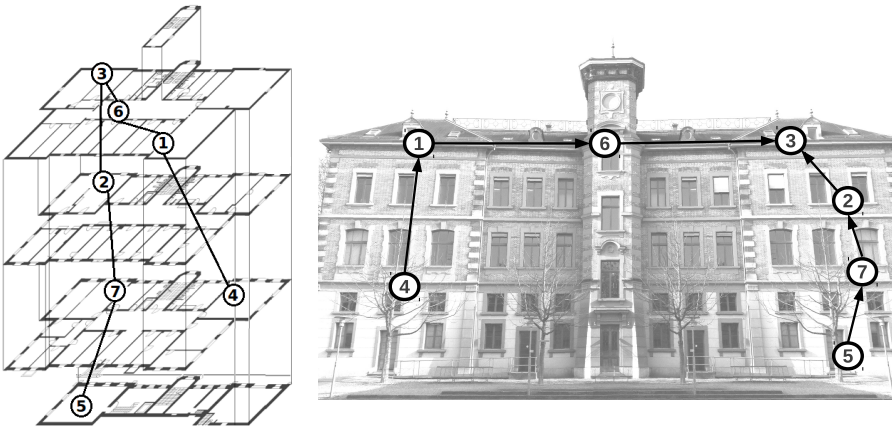


Fig. 8.12: Indoor Distributed Multi-Hop Scenario

collecting the results, cf. Chapter 3. The examined network topology is depicted in Figure 8.12. The nodes are distributed across four floors in one building, forming a V-shaped network with the sink node 3 in the top left corner. The evaluated scenario has been chosen to examine the impact of a somewhat elevated traffic in a network, but which is still far away from being congested: each sensor node except the sink node 3 generates and sends packets to its gateway node towards the sink node. The same transmission power settings (5 ticks $\approx$ -12.5 dBm) are statically set for every link. Static routes are further set in the beginning of the experiment, the respective links are depicted in Figure 8.12. We evaluated 1000 packets generated on each node for each run, once with the ECCs set in a static and network-wide manner, and once with each of the three adaptive FEC selection strategies. The delay between two generated packets follows uniform random distribution between 5 and 7 seconds, hence exhibiting a mean packet generation interval of 6s. Since packets generated at nodes 5, 7, 1 and 4 are transmitted over multiple hops to the sink node 3, the total amount of transmissions within the network amounts to 12 transmissions every 6 seconds, or 2 transmissions per second. Since the raw transmission time of one packet and the subsequent ACK takes roughly 50 ms + 20 ms (depending on the utilized ECC), we obtain a channel utilization of roughly 14% across the entire testbed. With this level of channel utilization, interferences due to other ongoing transmissions are likely to occur, which may render the application of ECC to be a valuable countermeasure.

We investigated the error patterns in the multi-hop scenario in the same manner as in the single-hop indoor and outdoor link evaluation. 1-bit and 2-bit errors were again the most frequently occurring errors in the multi-hop case. Figure 8.13 depicts the histograms of the probability distribution of 0 to 10 errors of the packets arriving at nodes 1,2 and 6,7 when applying no ECC. The figure clearly shows that the number of errors and the error pattern differs from link to link. Packets arriving at node 6 and 7 generally contain more errors and are more likely to contain more than 1-bit and 2-bit errors than those arriving at nodes 1 and 2. The most

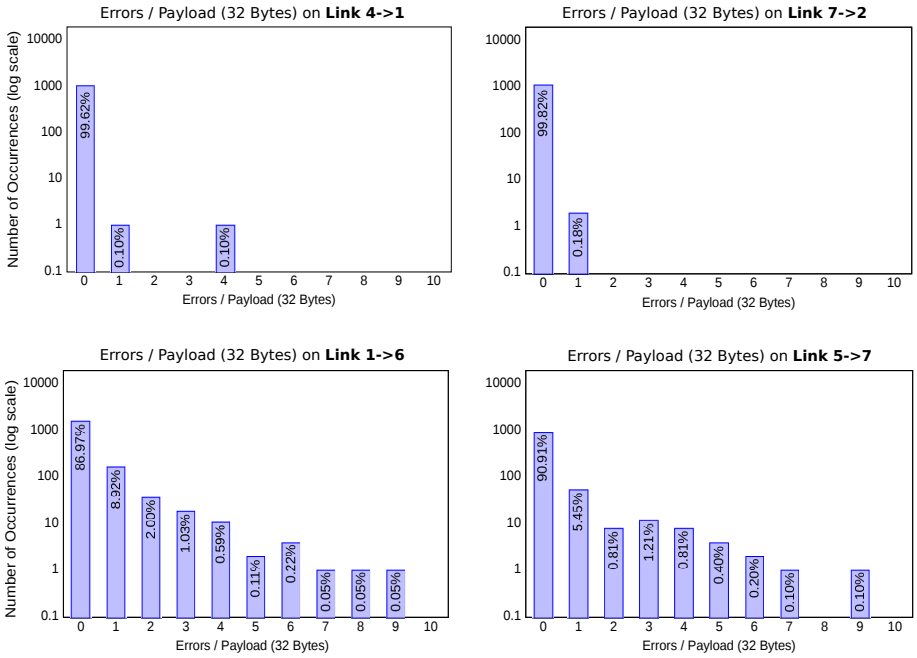


Fig. 8.13: Error Patterns: *Strong* Links (top) and *Weak* Links (bottom)

probable explanation for this observation is that the links $5 \rightarrow 7$ and $1 \rightarrow 6$ are prone to a higher absorption of the signal through walls and floors, since node 5 is in the basement of the building and has to penetrate a thicker floor than on the links $4 \rightarrow 1$ and $7 \rightarrow 2$ and $1 \rightarrow 6$ has to penetrate 5 concrete walls. The figure clearly shows that the encountered error patterns differ on each link, and that as a consequence, the decision whether an ECC should be applied should be taken on a per-link basis on the node itself. Clearly, pinpointing which links would turn out to be weak and error-prone and which ones would be strong and reliable would have been impossible in advance of network deployment, which renders the application of adaptive schemes particularly useful.

Figure 8.14 depicts the PDR bars of the examined ECC and the adaptive FEC approaches, leaving out the ECC codes that are never selected in the adaptive approaches (cf. Figure 8.2). The figure depicts for each examined setting the PDRs of the different links and the overall source-to-sink PDR. As one can clearly see in the top-left corner in the case of unencoded transmissions, the links $1 \rightarrow 6$ and $5 \rightarrow 7$ have a much lower success rate, which confirms the findings of Figure 8.13. When comparing with the case of unencoded transmissions (top left corner of Figure 8.14), one can clearly see that the application of FEC made transmissions along the error-prone links more reliable, especially on $5 \rightarrow 7$. The application of ECCs has clearly paid off with respect to alleviating the deteriorating impact of the lossy links $1 \rightarrow 6$ and $5 \rightarrow 7$: the improvement in the achieved PDR reaches up to 15% of the total generated packets (cf. Figure 8.14, DECTED and BCH).

Comparing the results of Hamming(7,4), DECTED(16,8) and the BCH-variants

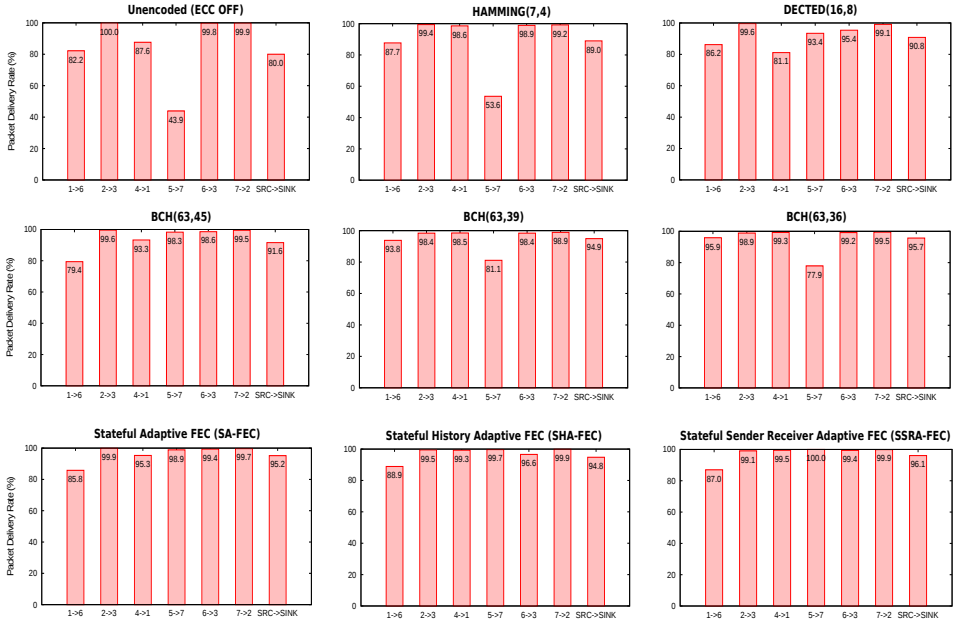


Fig. 8.14: PDRs per Link and Overall Source-to-Sink PDR

with the adaptive approaches, we can conclude that the adaptive FECs achieved astonishingly good results. The three strategies SA-FEC, SHA-FEC and SSRA-FEC outperformed almost every other static and network-wide setting of any of the implemented ECC codes. Since the adaptive schemes have only employed FEC on weak links and in periods of elevated BER, the majority of packets could be sent unencoded, which may also have led to fewer interference due to shorter transmission times, compared to static FEC settings. The major advantage of the adaptive approaches is the *on-demand* nature of using the correctional power of ECCs: with simple state-based concept, the adaptive approaches have managed to reach the same or better PDRs. Since the application of ECC comes at the cost of time and hence energy spent for encoding and decoding, ECCs should be limited to weak and error-prone links and/or time periods where the link quality suffers from deteriorating influences. Throughout the evaluation of this chapter, the obtained performance of the three approaches SA-FEC, SHA-FEC and SSRA-FEC with respect to the achieved PDR were in the same range, and hence determining a "winner" is difficult. Taking into account the energy consumption, the history-based approaches should probably be favored, since they are less prone to oscillation and do not immediately apply FEC after a single transmission failure.

8.5 Conclusions

In wireless networks, transmission errors appearing as bit flips in the received frames are more likely to occur than in traditional wireline networks, due to reasons ranging from signal absorption through concrete walls and floors, signal at-

tenuation due to long distances, reflection from obstacles or interference by other ongoing wireless transmissions. In this chapter, we have explored the potential of (adaptive) FEC techniques in the context of WSNs with lossy links. We have implemented eight different ECC codes in our library *libECC* and have proposed three run-time adaptive forward error correction schemes SA-FEC, SHA-FEC and SSRA-FEC, which react to deteriorating link quality by allocating the correctional power of ECC codes in an *on demand* manner.

We have analyzed the different codes in experiments conducted on single indoor and outdoor links and in a multi-hop network topology in a real-world sensor network testbed. We have gained valuable insight into the occurrence patterns of bit errors on indoor- and outdoor links, and different links in multi-hop networks. We chose to rely our entire investigation on real-world experiments, since simulation tools are inherently unreliable when dealing with wireless phenomena, which are as tightly linked to channel characteristics as FEC schemes. Phenomena such as random bit flips due to reflection effects, multipath propagation, or interference due to other concurrent transmissions are hard to model with network simulator environments, and have hence generally been put into question in the recent past, cf. *Kurkowski et al.* [105] and *Andel et al.* [11]. Our results conveyed that 1-bit and 2-bit errors across one transmitted frame were the most frequently encountered error patterns, but that frames containing more errors were also likely to occur on weak and lossy links. The occurrence pattern of transmission errors has hence turned out to be a rather *local* and *spatially* and also *temporally* variable issue, which in turn should be tackled on a per-link and not a network-wide basis.

By analyzing the results of the single and multi-hop experiments discussed in this chapter, we come to the conclusion that our proposed run-time adaptive FEC schemes have the following three major advantages:

- Adaptive FEC approaches qualify to cope with *timely variable* error rates, since they adapt the level of correctional power based on the success of the recent past transmissions. In case of timely correlated interferences (e.g., because of a device that is temporarily using the same or a near channel), adaptive FEC approaches can temporally switch to more powerful ECCs, and switch back again if the channel quality suffices to transmit packets unencoded.
- The occurrence pattern of transmission errors is a rather *local phenomenon*, which can differ heavily from link to link. Link qualities in turn are almost impossible to predict in full depth before network deployment. Our proposed adaptive FEC approaches adapt their correctional power used for each link individually by switching between simple and complex codes. Statically selecting the strongest ECC code in a network-wide manner may achieve a high success rate, but does also introduce the highest latencies, since transmissions across several hops require multiple encoding and decoding operations.
- By adaptively allocating the correctional power of different FEC codes, and adjusting the former to the currently encountered error rate, the effective payload data to be transmitted is minimized. In contrast to static FEC application, a ma-

jor share of packets is sent unencoded (cf. Figure 8.11), which also decreases the probability of interferences and collisions with other transmissions.

- The energy aspect favors the adaptive approaches: our evaluations of SA-FEC, SHA-FEC and SSRA-FEC have conveyed that when using these simple state-based adaptive FEC schemes, the major share of packets was still sent unencoded. The mechanisms succeeded well in deciding *when* and *where* to apply FEC in a totally distributed manner. In contrast to network-wide application of ECCs, precious resources for useless encoding and decoding operations on strong and reliable links can be saved, which likewise limits the overall energy overhead for both receivers and senders.

In this chapter, we have proposed and evaluated simple, yet effective state-based concepts of allocating the correctional power of Error Correcting Codes (ECCs) based on the encountered rate of successfully acknowledged packet transmissions over time. These concepts can generally be seen as a continuation of the state-based concepts of run-time *traffic-adaptive* medium access control mechanisms, which are discussed in Chapters 5 and 7, however, in a different problem domain. In both scenarios, WSN nodes attempt to *allocate* precious resources (i.e., battery power) in an *on-demand* manner when registering an increased requirement to do so, i.e., by observing Quality of Service metrics. In both evaluations, the simple state-based concepts have proved effective in increasing the network service characteristics (higher PDR, throughput, lower latency) while keeping the increase of energy consumption limited. Thanks to their simplicity, the discussed concepts further have the crucial advantage of exhibiting a low memory footprint, thereby satisfying further inherent WSN constraints.

By running the experiments overnight and on the weekends, we minimized the probability of irregular interferences caused by people working in the building, which could deteriorate the results of one particular parameter setting, and hence call the comparability of the individual FEC approaches' results into question. Our study hence particularly examines the potential of (adaptive) FEC mechanisms to cope with low signal-to-noise ratios, and high bit error rates due to multipath propagation, reflection or scattering effects, using a platform in the 804-940 MHz ISM frequency band. An evaluation of *libECC* and the proposed adaptive FEC strategies on IEEE 802.15.4-compliant radio chips would most probably convey different results, since bit error patterns in wireless communications often heavily depend on the modulation scheme and the radio frequency. Generalizations and conclusions drawn from this study can hence not safely be applied to other platforms, especially for those operating in a different frequency and using a different modulation.

In the subsequent Chapter 9, we present our contributions on reliable transport protocols operating on top of radio duty-cycled MAC layers in WSNs with different link qualities and the presence of interfering transmissions. We show that indeed, the concepts developed in 7 also make sense with respect to the transport layer performance, and that different duty-cycling algorithms have a significant impact on the overall TCP performance.

Chapter 9

TCP Optimizations for Wireless Sensor Networks

This chapter discusses our investigations and contributions on optimizations for the Transmission Control Protocol (TCP) in Wireless Sensor Networks (WSNs). As discussed in Chapter 2, the performance of TCP in wireless networks has been observed to perform poorly, especially across multiple hops. The reasons behind this performance degradation have been studied long before, in particular in the context of IEEE 802.11-based (mobile) ad hoc networks [31][80][68][15]. The reasons mainly lie in the unreliable nature of the wireless channel (higher bit error rates and packet loss), specific properties and interactions with the underlying wireless channel MAC protocols (exponential backoff mechanisms, hidden node and exposed node problem), and the design of the TCP congestion control mechanisms, which, due to historic reasons, erroneously interprets packet loss as an indication for network congestion.

In this chapter, we examine a series of TCP performance optimizations based on TCP segment caching and local retransmission strategies of intermediate nodes on the route of a TCP connection, and propose and evaluate own extensions to these strategies. We further particularly examine the impact of different radio duty-cycling energy-efficient MAC (E^2-MAC) protocols on the end-to-end TCP performance. The contributions of this chapter relate to the superordinate topic of the thesis, which deals with adaptive communication architectures, by the adaptive nature of the distributed caching and retransmission strategies presented in the following. Based on the perception of signals of service degradation by intermediate nodes (i.e., the observation of packet loss by a timed-out RTO, or the observation of the channel remaining idle for an unusual long time), the mechanisms proposed in this chapter attempt to cope with the adverse circumstances of low-power and error-prone wireless links in WSNs. By immediately reacting to timely and spatially variable packet loss with dynamic allocation of precious resources (i.e., local retransmissions), yet in a totally distributed manner, our proposed strategies attempt to prevent further TCP service degradation.

Unfortunately, it proved to be impossible to fully link our investigations on the

traffic-adaptive MAC protocol MaxMAC presented in Chapter 7 with the TCP optimizations contained in this chapter. The employed ScatterWeb^2 Operating System used for prototyping MaxMAC unfortunately does not offer an integrated µIP stack. Due to the tight integration of the MaxMAC implementation with the ScatterWeb^2 OS and the byte-level oriented radio, porting the ScatterWeb^2 MaxMAC implementation to Contiki on TelosB nodes turned out to be impracticable.

The results of this investigation were published in [96][26]. In the following, the chapter is structured into four sections. In Section 9.1, we motivate the need for robust operation of TCP on top of radio duty-cycled E^2-MAC protocols in WSNs. In Section 9.2, we present various suggestions for TCP optimization across multiple hops, which we then examine in Section 9.3 in a series of real-world testbed-based experiments. Section 9.4 concludes the chapter.

9.1 Motivation

The TCP/IP protocol suite is the undisputed standard for communication in the Internet today. With the emergence of distributed systems and small networked embedded devices, such as WSNs, researchers have successfully downsized and simplified TCP/IP networking stacks to be capable of interoperating with "normal" and fully RFC1122 [23]-compliant TCP/IP implementations on hosts in the Internet. With the development of the µIP Stack [45], a large number of devices on different platforms can nowadays be directly accessed over TCP/IP with commonly used tools and applications (e.g., Telnet [148] or SMTP [145]), instead of using protocol proxies or other kinds of middle-boxes in between the WSN nodes and the Internet. Furthermore, reliable transport is increasingly important in WSNs for message exchanges where random packet loss e.g., when disseminating program code, configuration updates or highly important status notifications. Among the issues discussed in Section 2.6 of Chapter 2, which especially hold for IEEE 802.11-based networks, the application of TCP/IP in WSNs faces additional challenges, which are related to the inherent WSN constraints, i.e., the limited memory and computational power, and the energy restrictions. In order to prolong WSN lifetimes to more than a few days, the application of radio duty-cycling E^2-MAC protocols has become common sense in today's WSN communities.

Yet, except for the implementation of *bulk-transfer* schemes, the issue of optimizing TCP in the context of WSNs with lossy links has not attracted too much attention in the recent past. *Duquennoy et al.* [52] have recently modified the µIP Stack to implement such a *bulk-transfer* protocol on top of ContikiMAC. In [52], large bursts of TCP segments are sent in a hop-by-hop manner towards the other TCP endpoint, assuming that each node can buffer significant amounts of data (~ 1 MB) on external flash memory. However, the approach significantly increases the end-to-end latency, which qualifies it for transport of large data portions (e.g., code or configuration updates), but which disqualifies it for applications with tight latency requirements (e.g., manual real-time interaction with remote sensors).

The studies DTC [46] and TSS [24], which are discussed in detail in Sections 2.6.3 and 2.6.4 of Chapter 2, propose mechanisms based on packet inspection in intermediate nodes, i.e., distributed TCP caching in [46] and, in addition, TCP acknowledgement regeneration in [24]. However, both studies base upon evaluations on network simulations with rather simple channel assumptions, and rely on having an energy-unconstrained CSMA/CA on the MAC layer, in order to overhear neighboring node's transmissions. Thus, the implications of radio duty-cycling E^2-MAC protocols on the performance of distributed caching and local retransmission schemes proposed in [46][24] have generally not yet been studied.

9.2 TCP Performance Optimizations

The starting point of our investigations is formed by the TCP segment caching strategy proposed in DTC and TSS, which perform so-called *local retransmissions* of TCP segments and *regeneration* of TCP acknowledgements. Based on ideas of these two studies, we iteratively designed, tested, and combined a number of TCP performance optimizations, which we discuss in the following. Our optimizations are to a large extent independent from each other, consist in either new functionalities or modification of previous suggestions. We implemented all our modifications and TCP optimizations in a modular manner into our so-called *Caching and Congestion Control (cctrl)* module. The module can be integrated into the µIP stack [45] of the Contiki OS [47], which serves as our main platform for implementation and evaluation in this chapter, yet remaining transparent to the underlying MAC layer and the application layer above. The transparency to the underlying network stack was a major design goal of the *cctrl* module, since it allows for using all its implemented features on top of every MAC protocol layer currently available for the Contiki OS. The only prerequisite of the *cctrl* module to effectively interact with the flow of TCP packets is to have *symmetric routes* in the WSN, such that TCP acknowledgements are sent across the same paths back to the sender as the corresponding TCP data segments. Since this is not ascertained in the *default* configuration of Contiki, we implemented simple commands to statically configure routes of the µIP layer at run-time.

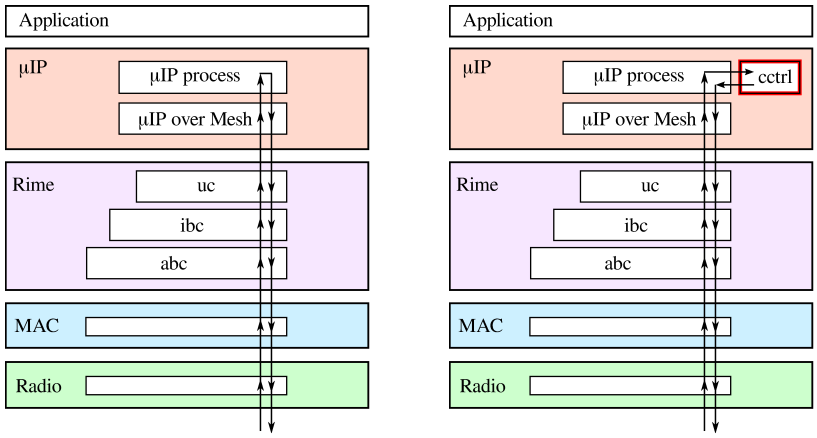
The remainder of this section is structured as follows: we describe the integration of our proposed *Caching and Congestion Control (cctrl)* module into the Contiki OS network stack in Section 9.2.1. We then discuss the fundamentals of the initial caching strategy of the *cctrl* module in Section 9.2.2, which applies the basic distributed caching and local retransmission features from DTC [46] and TSS [24]. Sections 9.2.3 and 9.2.4 then introduce our proposed extensions, which rely on the integration of MAC-layer specific information about the current channel utilization (9.2.3), or the introduction of multiple concurrent TCP sub-connections to overcome the µIP's limitation of one single segment *in flight* per socket (9.2.4).

9.2.1 The Caching and Congestion Control (cctrl) Module

In order to locally cache TCP segments in intermediate nodes in between two TCP endpoints, the *cctrl* module has to be aware of all TCP packets that are forwarded. Figure 9.1 depicts a packet traversing the different layers of the Contiki network stack - Figure 9.1(a) the unmodified stack, and Figure 9.1(b) the stack with our *cctrl* module integrated. A TCP packet, encapsulated in a Rime [49] packet, is received by the radio and passed to the MAC layer, which copies it to Rime's packet buffer. From here, it is processed by the various Rime modules responsible for sending and receiving unicast traffic. Rime is a collection of different node-to-node communication services, with simple services providing basic features, and more complex services building on top of them, cf. Section 2.1.6 of Chapter 2.

The Rime layer then copies the payload to the µIP layer's packet buffer, where the *µIP over mesh* module is notified. *µIP over mesh* implements the routing functionality in networks across multiple hops, and finds the gateway for a given target destination. Throughout all our experiments, however, we altered this layer to set statically configured routes, since the mechanisms outlined in this chapter rely on the assumption of symmetric routes. As the node in Figure 9.1 is an intermediate node and not the final recipient of the packet, the packet is sent out again, traversing the aforementioned layers in reversed order. The *cctrl* module intercepts the packet flow right before the packet is again passed to the *µIP over mesh* module, cf. Figure 9.1. One single file of the Contiki µIP stack has to be slightly altered, such that the *cctrl* module gets access to process the outgoing packet while it resides in the IP buffer. We intentionally intercept outbound packets instead of inbound packets, since at this stage packets have already been processed and validated.

To be able to cache TCP segments, the *cctrl* module needs to allocate memory. The allocated buffer has to be large enough to hold at least the content of two TCP/IP packets per observable connection, one for each direction. Since at least



(a) Unmodified Contiki Network Stack (b) Contiki Network Stack with *cctrl*

Fig. 9.1: Packet traversing Layers in Contiki Network Stack - with and without *cctrl*

one host is assumed to be running the µIP stack, which is restrained to only one single unacknowledged segment per TCP connection, the *cctrl* module allocates two times the size of µIP's packet buffer per observed TCP connection.

9.2.2 Initial Strategy: Segment Caching and Local Retransmissions

In our initial design of the *cctrl* module, we implemented the basic distributed caching features from DTC [46] and TSS [24] that are *independent* from the underlying MAC protocol. We generally avoided to rely on the overhearing assumption and to tightly intertwine our *cctrl* module with the MAC layer, since we intended to examine the former with all available MAC layers of the Contiki OS, in particular with the radio duty-cycled E^2-MAC protocols. The TCP segment caching and local retransmission strategy of our initial design of the *cctrl* module is depicted in Figure 9.2, and will be denoted as *initial cctrl* hereafter in the graphs of Section 9.3.

- In each intermediate node between two TCP endpoints, the *cctrl* module processes each IP packet received and forwarded. IP packets without a TCP payload (e.g., UDP packets) are ignored. Only packets with TCP payload data are cached. The *cctrl* module therefore copies the entire content of the µIP buffer to one of its empty buffer slots, and schedules a retransmission timer. In case of a TCP acknowledgement, only the TCP/IP header is cached, and no retransmission timer is scheduled. Caching is omitted for out-of-sequence packets and retransmissions, such that the connection status of the *cctrl* module always holds valid sequence and acknowledgement numbers.
- When the retransmission timer of a forwarded TCP segment expires before a TCP acknowledgment has returned, the *cctrl* module assumes that the segment has been lost, releases it from the cache and sends it out again, as displayed with label (a) in Figure 9.2, with node C initiating a local retransmission.
- Each TCP segment is checked according to the state of the TCP connection it belongs to. In case the current packet is a TCP acknowledgement of a cached TCP segment, the *cctrl* module checks whether the current segment's acknowledgement number is greater than the cached segment's sequence number, and removes this segment's cache entry if this is the case.
- If the packet is a retransmission of a TCP segment, for which a TCP acknowl-

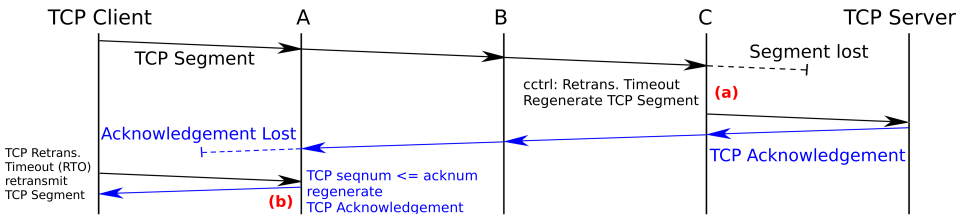


Fig. 9.2: Local TCP Retransmission (a) and TCP Acknowledgement Regeneration (b)

edgment has already been received, it is discarded and a TCP acknowledgement is sent towards the packet's sender. This is displayed with label (b) in Figure 9.2, where node A responds to a TCP retransmission by regenerating the previously received TCP acknowledgement. The regeneration of TCP acknowledgements is not linked to a timer, in contrast to the approach followed in TSS [24]. It is only triggered when the aforementioned condition of an already acknowledged TCP segment arriving at an intermediate node is encountered.

The *cctrl* module distinguishes between two reasons for packet loss that make a retransmission necessary: a data packet containing a TCP data segment that is lost on its way from the sender towards the recipient, or a TCP acknowledgment getting lost on the same path in the opposite direction. In contrast to DTC or TSS, which overhear the next node's transmissions in order to obtain an implicit acknowledgement for the success of their own transmission, we assume that packet losses on the link layer are not detectable on the IP layer, because E^2-MAC protocols generally do not support continuous overhearing. The retransmission scheduling is similar to that of DTC and TSS: for each cached TCP segment, *cctrl* schedules a timer, upon expiry of which a retransmission is initiated, i.e. in the case when it takes considerably more time than usual for the TCP acknowledgment to arrive. In each intermediate node participating in a TCP connection, the *cctrl* module henceforth calculates for each TCP connection the TCP retransmission timeout (RTO), using the original TCP Exponentially Weighted Moving Average (EWMA) filter [147]:

$$RTT_{est_{new}} = RTT_{est} \times \alpha + RTT_{est} \times (1 - \alpha) \qquad \text{(EWMA-RTT)}$$

The initial value of the retransmission timeout RTT_{est} was set to 2 seconds, which proved to be high enough to ensure that the first cached segment successfully gets acknowledged by the destination, and no intermediate node triggers an early retransmission for our examined MAC protocols and route lengths. When a cached, but not yet retransmitted segment is acknowledged, the *cctrl* module's round trip time estimation RTT_{est} to the receiver of the cached segment is updated with $\alpha = 0.25$. The retransmission timeout is then set as $t_{rto} = n * rtt$, with $n = 3$. We experimentally determined the values $\alpha = 0.25$ and $n = 3$ in a preliminary

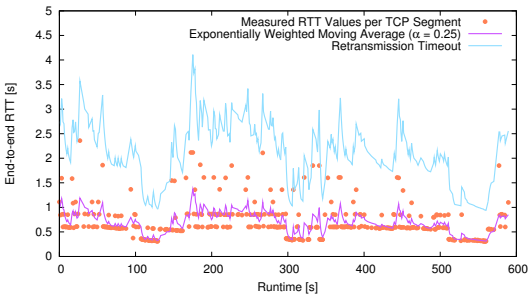


Fig. 9.3: X-MAC Example Run: RTT and TCP Retransmission Timeout (RTO)

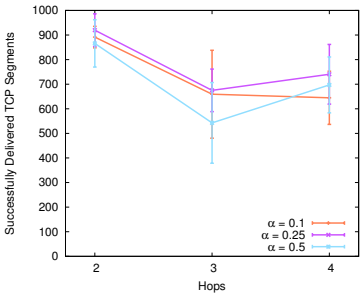


Fig. 9.4: Throughput for $\alpha \in \{0.1, 0.25, 0.5\}$

investigation with a series of experiments across 2, 3 and 4 hops. Figure 9.3 visualizes one run of such an experiment, where a node continuously sends TCP segments across 2 hops during 10 minutes using X-MAC. The figure depicts the single RTT measurements, the resulting RTT estimation rtt obtained with applying the EWMA filter (cf. equation EWMA-RTT), and the retransmission timeout t_{rto} set to $3 * RTT_{est}$. The figure conveys a rather high variability of the RTT measurements in the upwards direction. Most values are between 0.5s and 1s, but there is a rather large portion of outliers up to the value of 2.5 seconds. With choosing the RTT multiplier $n = 3$ for the calculation of the RTO, we minimize the probability of triggering early retransmissions. When using the value of $n = 1.5$ proposed in TSS [24], early retransmissions lead to interferences and collisions with returning TCP acknowledgments on their way back to the sender, and generally deteriorate the end-to-end TCP throughput rather than improving it.

Another series of preliminary experiments then supported the choice of the values $n = 3$ and $\alpha = 0.25$ as well: Figure 9.4 depicts the total number of transmitted TCP segments during 10 minutes, running our initial TCP caching strategy with X-MAC and values of $\alpha \in \{0.1, 0.25, 0.5\}$. As Figure 9.4 clearly conveys, the best results were achieved with $\alpha = 0.25$. We therefore sticked to $n = 3$ and $\alpha = 0.25$ for the entirety of our evaluations.

To prevent the *cctrl* cache to run full of stale TCP connection entries, maintenance timers periodically check each observed connection for their liveness. These timers are initially set to expire after 60 seconds, but are reset every time a TCP packet from their TCP connections is forwarded. Upon expiry of a maintenance timer, the corresponding connection information and any remaining cached segments entries are removed. The same is done when forwarding TCP FIN or TCP RST packets, which indicate that the TCP connection has either been terminated or timed out.

Multiple Retransmissions and Duplicate Segment Dropping

In DTC and TSS, each intermediate node removes the segment from the cache after it has retransmitted it once. This strategy is depicted in Figure 9.5, where nodes C and B both remove the cached segment from the *cctrl* cache after their first retransmission. In preliminary tests, we have experienced that limiting the number of retransmissions to only one single attempt can be disadvantageous. When the first transmission of a packet is lost and, shortly after, the retransmission as well, the "burden" of retransmitting the segment again is moved towards the sender, rendering successful reception by the TCP server less and less likely. This is illustrated in Figure 9.5, where node C's original transmission and the subsequent retransmission both get lost. Such behavior is rather probable in WSNs, where packet loss rates tend to fluctuate over time, e.g., because of timely correlated transmission attempts and collisions, or environmental influences, e.g. weather conditions or mobile obstacles (e.g., animals). If each node can only retransmit once, the burden to successfully retransmit is moved further and further away from the targeted receiver, rendering the success less and less probable.

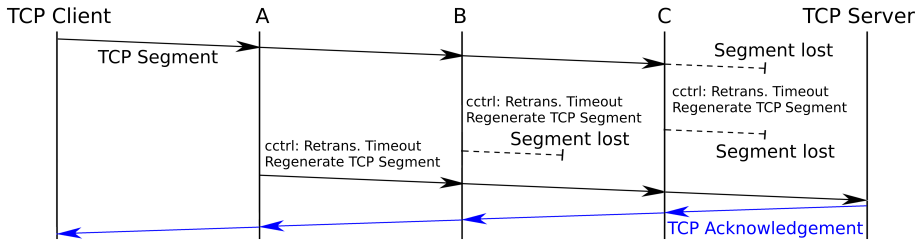


Fig. 9.5: Failing Retransmissions move the Burden towards the Sender

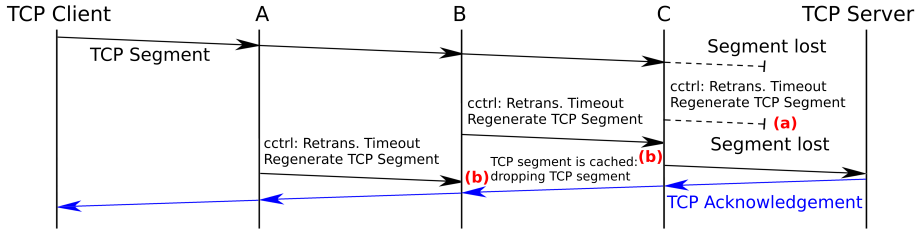


Fig. 9.6: Multiple Retransmissions (a) and Duplicate Segment Dropping (b)

We hence modified our caching strategy by increasing the maximum number of retransmissions. After a first retransmission, the segment remains cached and the retransmission timer gets rescheduled with the same value of $t_{rto} = 3 * RTT_{est}$, as illustrated in Figure 9.6 with node C retransmitting again after the first *two* transmission attempts have been lost. We limited the total amount of TCP segment retransmissions to 3 attempts. Upon failure of these 3 attempts, the TCP segment is removed from the cache and dropped definitively. However, the drawback of increasing the retransmission limit is that after a packet loss, all intermediate nodes repeatedly retransmit the segment, which increases the channel utilization in the network with often redundant retransmissions. Our countermeasure for this problem is a slight modification of the TCP segment forwarding policy depicted in Figure 9.6 labeled (b), according to which a retransmission of a cached segment is dropped, in order to reduce the load on the subsequent links. We will refer to the segment caching and retransmission strategy outlined in this section, including the aforementioned retransmission limit set to 3, as well as the duplicate segment dropping policy, as the *initial cctrl* approach in the evaluation of Section 9.3.

9.2.3 Channel Activity Monitoring

Numerous WSN protocols and frameworks have yet proposed to take advantage of the broadcast nature of omnidirectional wireless transmissions, in order to gain additional information about ongoing transmissions in the vicinity. DTC and TSS use overhearing of forwarded packets of their neighboring nodes as implicit acknowledgment for successful packet reception. In CODA [180], the channel conditions of the recent past and the amount of buffered packets in the TinyOS internal send queue are used to calculate the so-called *channel loading* factor, which is used as

an indicator for congestion. Most features of CODA, however, again rely on the assumption of energy-unconstrained CSMA, similarly as DTC and TSS.

When a radio duty-cycling E^2-MAC protocol is used on the MAC layer, continuous overhearing of every transmission of the neighboring nodes is not possible. Transmissions can only be overheard when the radio is currently turned on, e.g. because the E^2-MAC protocol has scheduled a wake-up of the radio. However, even the coincidental reception of frame transmissions of other nodes can provide valuable information. With preamble-strobing MAC protocols, e.g., X-MAC, the overhearing of a strobe indicates that there are currently ongoing transmissions in the vicinity. A large amount of overheard packets over the recent past always indicates a situation in which it would be beneficial for a node to withhold a scheduled transmission to avoid further collisions, independent from the MAC layer that is used. We therefore designed a solution that a) remains independent from the MAC protocol, but that b) still permits to feed information about the current channel utilization and channel conditions to the *cctrl* module.

The MAC Proxy Module

In order to get access to the MAC layer, where overheard packets are dropped and not passed further in the Contiki network stack, we implemented a simple hook for the *cctrl* module to the MAC protocol's packet buffer, however, keeping the MAC protocol completely unmodified and replaceable. Our solution consists in the so-called *MAC Proxy* module, which implements the MAC layer interface of the Contiki OS, but does not provide any functionality on its own, except for notifying the *cctrl* module upon reception of every packet. The MAC protocol proxy module initializes a real MAC protocol layer of the Contiki OS (e.g., the NullMAC, X-MAC, or LPP layer), and simply forwards every function call. Figure 9.7 illustrates the modified packet reception workflow with the newly introduced MAC Proxy.

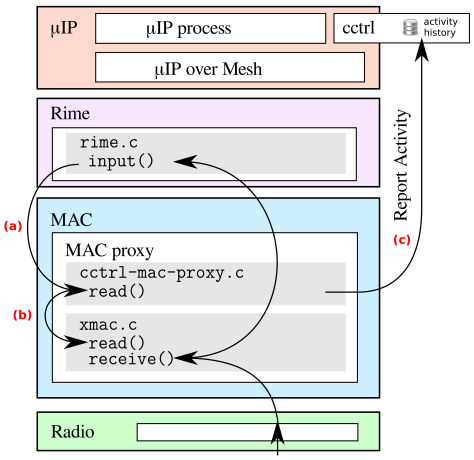


Fig. 9.7: Processing Overheard Packet using MAC Proxy Module

When a frame is received by the radio in Contiki, the radio driver calls the MAC layer's `receive()` function. The MAC then informs the Rime layer about this event by calling `input()`. Rime then calls the MAC protocol's `read()` function to start the actual parsing. Using our MAC Proxy situated in between the Rime layer and the *real* MAC protocol (e.g, X-MAC in Figure 9.7), Rime calls the `read()` function of the MAC Proxy instead of that of the *real* MAC protocol, cf. Figure 9.7 label (a). The MAC proxy forwards this call to the actual real MAC protocol, cf. Figure 9.7 label (b), which parses the packet and copies the received data to the Rime buffer. In case of an error, e.g., a CRC checksum mismatch, or a wrong target address, the return value is zero and the packet is dropped. Even in this case, though, the MAC protocol proxy becomes aware of the overheard but corrupted packet, and notifies the *cctrl* module about the overheard packet by storing a timestamp into its newly introduced *activity history*, c.f. Figure 9.7 label (c). This data structure stores the timestamps of the 20 most recent activity notifications received from the MAC proxy. It is used to calculate the current *channel activity level*, which we define as the amount of overheard packets registered by the MAC proxy that are not older than one average RTT estimation RTT_{est} of the observed TCP connection. Relating the activity level calculation to RTT_{est} compensates for the large differences among the MAC protocols' latencies. Static values would have to be inconveniently configured for each MAC protocol.

Idle Channel Periods

In a preliminary evaluation, we investigated whether the calculation of the activity levels has any informative value, especially when overhearing is only possible to a minor extent due to the application of radio duty-cycling E^2-MAC protocols. We

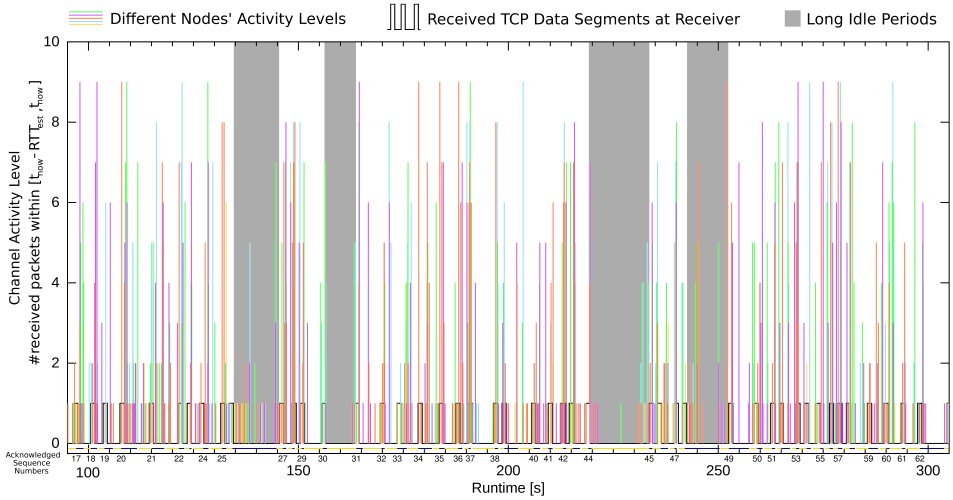


Fig. 9.8: Different Nodes' Activity Levels with X-MAC vs. Experiment Time. Dashed Line indicates Traveling Time for Data Segments (*yellow*) and Acknowledgments (*blue*).

evaluated the obtained activity values in a scenario using X-MAC, where one TCP sender at the beginning of a 7-nodes linear chain sends TCP segments to the receiver at the end of the chain. Five nodes with *cctrl* modules hence forward the TCP data segments and TCP acknowledgements between these two nodes. Figure 9.8 depicts the activity levels registered by the *cctrl* modules of the five intermediate nodes versus the experiment time. Each node is represented by one specific color, the reception of data packets at the end node and the corresponding sequence number are displayed along the x-axis. During most of the time, all nodes register rather high activity levels: 6-8 packets are overheard within each node's RTT_{est} on average, since with X-MAC sampling the channel each 125 ms per default, chances are high that some preamble strobes of currently ongoing transmissions are overheard by non-targeted nodes. Between t=100s-130s and t=250s-300s, the flow of TCP segments and acknowledgements is continuous. However, some TCP segments (e.g., sequence numbers 27, 31, 45, 47 etc.) need significantly longer to be delivered. During these time periods (e.g., t=150-160s), little or no channel activity is registered by all the nodes. We investigated the problem for these interruptions in the TCP flow, during which precious bandwidth remained unallocated.

An in-depth analysis of the trace files revealed that there are two common problems that caused these rather long *idle periods*, which are visualized in Figures 9.9 and 9.10. The first problem consists in the loss of a TCP segment on one of the first hops in the connection. In such a case, the segment has not yet been cached by many intermediate nodes (e.g., only one in Figure 9.9), and hence, the network

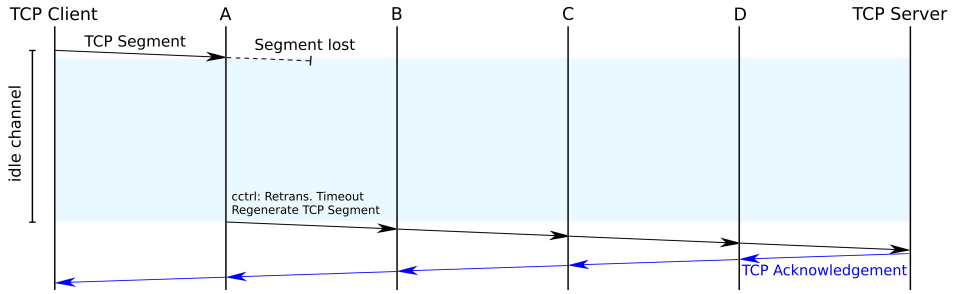


Fig. 9.9: Packet Loss close to the Sender results in long idle Period

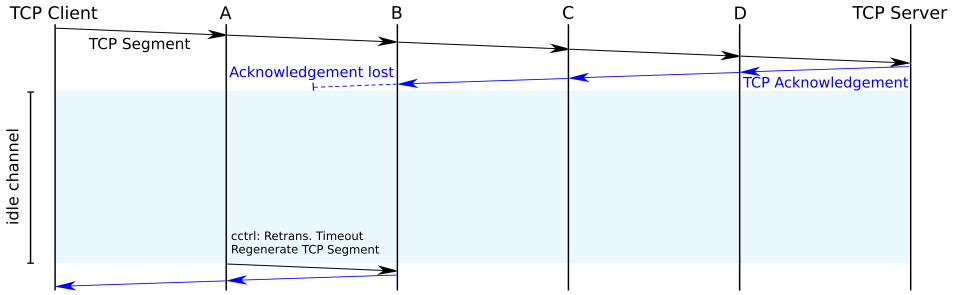


Fig. 9.10: TCP Acknowledgement lost close to its Destination results in long idle Period

remains idle for at least $t_{rto} = 3 \times RTT_{est}$. During this potentially long idle period, all nodes in the chain have to wait for the RTO to occur. A second frequently encountered problem occurs when a TCP acknowledgment is lost close to its final destination. The retransmission of the TCP acknowledgement can then only be triggered by the reception of a retransmitted data packet, which has to come from one of the first nodes in the route, as depicted in Figure 9.10, because all other intermediate nodes traversed by the TCP acknowledgement have already emptied their cache. The more nodes the TCP acknowledgement has already passed, the higher the resulting idle waiting time.

Activity Dependent Early Retransmissions

We searched for a means to exploit the additional channel knowledge gained with the employed MAC protocol proxy and channel activity history, in order to resolve or alleviate the discovered problems of the long idle periods and further improve the end-to-end TCP throughput. According to our observation that the activity values of all the nodes equal to zero in the discovered situations of an idle waiting period (cf. Figure 9.8), we altered the *cctrl* module's retransmission mechanism to retransmit *earlier* when the channel was found idle for a long time. The retransmission timer of cached TCP segments was hence split into two parts: $t_{rto} = t_{rto1} + t_{rto2}$. The first timer times out at $\frac{2}{3}$ of the usual RTT estimation value RTT_{est}, hence $t_{rto1} = 3 \times RTT_{est} \times \frac{2}{3}$, and the second timer $t_{rto2} = 3 \times RTT_{est} \times \frac{1}{3}$. When the first retransmission timer expires, the *cctrl* module checks its activity history, and initiates an early retransmission given that the activity level equals zero. Otherwise the retransmission is deferred again by the second timer t_{rto2}. With this retransmission strategy, we targeted at reducing the occurrence probability and the duration of the discovered long *idle periods*.

Since the value of RTT_{est} decreases towards the nodes *closer* to the destination of the TCP data segments, the *channel activity level* value (calculated as the number of packet receptions within $[t_{now} - RTT_{est}, t_{now}]$) decreases as well, and is more likely to equal zero than at the beginning of the chain. Therefore, the outlined strategy triggers the retransmissions of the nodes closer to the destination earlier than those at the beginning of the node chain, given that they have yet received the initial transmission of the segment. This generally renders re-transmissions close to the destination more likely, which is a desirable property. The absolute value of the *activity level* has no particular deeper meaning. Our outlined retransmission strategy is only triggered when its value is zero, a situation in which it is probable that the channel has not been used and has hence been left idle for $2 \times RTT_{est}$.

9.2.4 Multiple Connections

Since the Contiki µIP stack only allows the transmission of one unacknowledged TCP data segment at any time per TCP connection, precious bandwidth could probably remain unallocated, especially on long routes, where transmissions on one

end of the route would not necessarily interfere with transmissions on the far other end. We searched for a means to spatially reuse the wireless channel and to allow the transmission of multiple segments *in flight*. However, sticking to our initially described design decisions, we decided to keep the *cctrl* module modular, independent from the MAC layer and from modifications within the established µIP stack itself. We hence designed a simple solution that simultaneously establishes multiple TCP connections between TCP client (sender) and TCP server (receiver). This allows an application to send out a new data packet over a second TCP connection, although the previously transmitted packet has not yet been acknowledged, permitting a maximum of two TCP segments or ACKs *in flight*. This solution could also alleviate the impact of the long *idle periods* noticed beforehand, since after a packet loss of one TCP connection and the subsequent retransmission timeout, the second TCP connection could still operate and use the available bandwidth.

The effective implications of this approach were yet unforeseeable at the time of designing it: instead of an improvement of throughput, it could also result in a deterioration, since more TCP segments being transmitted could also lead to congestive situations and an increasing number of collisions. We modified the client application in the application layer to open exactly *two* TCP connections to the same server instead of only a single one, and to let the application send two TCP data segments at a time. Fragments arriving in wrong order are rearranged with a buffer holding a couple of segments in the receiver's *cctrl* module, as depicted in Figure 9.11 in the network stack of the TCP server. Our implementation of this strategy remains fully transparent to the application *and* the µIP stack. Whenever the TCP client sends data, it is tunneled by the *cctrl* module over either of the two connections, without the µIP stack being aware that the two opened connections actually belong to the same socket on the application layer.

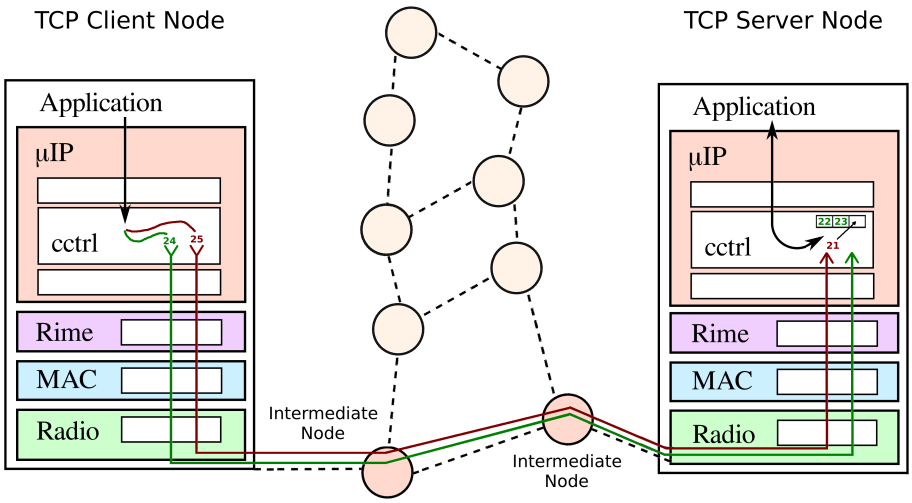


Fig. 9.11: Multiple TCP Connections between TCP Endpoints

9.3 Experimental Evaluation

For all the subsequent experiments and evaluations of this section, we used our distributed indoor testbed facilities operated by our testbed management architecture TARWIS [92][84], which is discussed in detail in Chapter 3. TARWIS allowed us to repeatedly run a large number of experiments without having to be physically present and without any continuous interaction with the testbed, which massively facilitated and expedited experimentation. Section 9.3.1 discusses the experiment resources, the topology setup and further experiment settings, which are followed by the discussion of the experiment results in Sections 9.3.2 and 9.3.3.

9.3.1 Experiment Setup

The two topologies used for experimentation are depicted in Figure 9.12. We used a subset of the currently deployed 40 TelosB [144] sensor nodes. The used topology is located in the Institute of Computer Science and Mathematics' main building at Neubrückstrasse 10 in Bern, cf. Section 3.4 of Chapter 3. More details about the TelosB platform can be found in Section 2.1.5 of Chapter 2.

We examined the following two different experiment scenarios, each with different lengths of the employed routes: in the first scenario, which is discussed in Section 9.3.2, data was transmitted over 2, 3, 4, 5 and 6 hops on one single path, depicted as the *red* route in Figure 9.12. In each of these single route experiment configurations, node 1 formed the TCP server (the receiver of the TCP data segments), and nodes 5, 6, 7, 10, 13 were the TCP clients (the senders of the TCP data segments) for the experiments with the different route lengths.

In the second scenario, which is discussed in Section 9.3.3, we examined the case where TCP traffic is being handled across two routes at the same time, with the

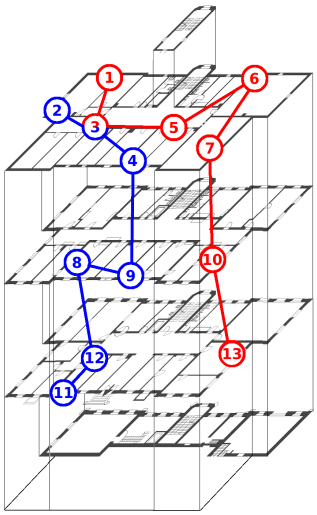


Fig. 9.12: Single Route Scenario (red) and Cross Traffic Scenario (red and blue)

routes being intertwined, i.e., sharing a common node (cf. node 3 in Figure 9.12). Again, we examined 5 different configurations, with the two routes being either 2,3,4,5 or 6 hops long. In each configuration, nodes 1 and 2 formed the TCP server. The TCP clients varied, depending on the route length, from nodes 5-13 for the red route and 4-11 for the blue route. As one can clearly see in Figure 9.12, nodes are located in different rooms of the building, with the configured route spanning across three floors. Nodes hence communicate across concrete walls and floors. The links of the employed routes exhibit a rather high success rate in case of no other ongoing transmissions ($\geq 75\%$). Besides the links displayed in the figure, some close node pairs physically were within each other's transmission range (e.g., nodes 5 and 7). However, for many node pairs, direct communication failed because of the signal attenuation from obstacles (e.g., walls, floors) and/or the distance (e.g., 1, 6, 13 and 11 were out of each other's transmission range).

We used each of the four currently available Contiki OS MAC protocol layers, which are discussed in more detail in Sections 2.4 and 2.1.6 of Chapter 2. By running all the experiments with four different wireless channel MAC protocols, among them three radio duty-cycling E^2-MAC protocols, we thoroughly investigate the impact of the MAC layer on the end-to-end performance of the distributed caching schemes proposed by DTC [46] and TSS [24]. To the best of our knowledge, distributed caching approaches applied in WSNs have not yet been studied by means of real-world prototypes. In particular, no existing study has yet investigated the impact of E^2-MAC protocols on these mechanisms. The examined Contiki OS MAC layers taken from the Contiki OS' source repository are the following:

- the *NullMAC* layer, which, combined with the Contiki CSMA layer operates as simple energy-unconstrained CSMA with a random backoff contention,
- the prominent *X-MAC* [25] protocol layer applying asynchronous preamble sampling and preamble strobing,
- the *ContikiMAC* [48] layer, which merges features from a range of asynchronous preamble-sampling E^2-MAC protocols, and which has become the new default MAC layer of the current Contiki OS v.2.5, and
- the receiver-initiated *Low Power Probing (LPP)* [130] layer, with which every node periodically sends out beacons to indicate reception readiness.

We chose every experiment run to last exactly 10 minutes, during which the TCP clients on one end of the route tried to send as many segments as possible to the TCP servers on the other end. We kept the size of the individual packets constant such that comparability among the experiment runs is ensured. All TCP data packets contained a 16 byte character string as payload. Including the TCP/IP and Rime headers, the radio had to transmit 79 bytes per data frame. A TCP acknowledgment, as transmitted by the servers in response to a data packet, contains only 63 bytes in total, since it does not contain any TCP payload. Throughout the entire experiment duration, the involved nodes printed their stats to their serial interfaces, which were then collected using the TARWIS management system.

All experiments were run *with* and *without* our proposed *cctrl* module and the extensions proposed in Sections 9.2.3 and 9.2.4. The results of the experiment runs obtained without the *cctrl* module, hence with the same application sending TCP packets, but running an unmodified Contiki OS networking stack, are henceforth referred-to as *unmodified* in the subsequent figures. In order to reduce the impact of environmental impacts on the experiment results, all experiments were run over night or during weekends, when none or only few people are expected to be present in the building and, hence, the channel circumstances are comparable to a large extent. Since the frequency band of the TelosB nodes is license-free, there are many consumer electronics in the same or a near frequency range, e.g., IEEE 802.11-based devices, bluetooth devices, cordless headphones or microwave ovens. We generally experienced more stable results at non-working hours.

Each run was repeated 15 times, in order to obtain a meaningful data set from which stable statistical measures could be calculated (i.e., mean and standard deviation). The figures of this section all depict the mean values of the 15 runs of each configuration, and the standard deviation bars depicting $\pm\sigma$. In total, the data presented in this chapter was acquired throughout 2500 experiment runs, an equivalent of 425 hours experiment time.

9.3.2 Single Route Scenario

Figure 9.13 depicts the number of successfully transmitted TCP segments of the four examined MAC layers, dependent on the number of hops. The graphs labeled *initial cctrl* refer to our initial design of the *cctrl* module discussed in Section 9.2.2, which implements the basic features of DTC and TSS, but without the extensions of Sections 9.2.3 and 9.2.4 (*channel activity monitoring* and *multiple connections*).

Initial Strategy: Segment Caching and Local Retransmissions

Figure 9.13 conveys that NullMAC clearly benefits from the caching and retransmission mechanisms introduced with the *initial cctrl* design, most notably when data travels across long routes consisting of 5 and more hops, where it reaches almost twice the throughput of the *unmodified* Contiki network stack. For the shorter routes, the result is less distinctive, but the application of local retransmissions still tends to be an improvement. For X-MAC, throughput remained more or less in the same range with the *initial cctrl* approach as with *unmodified* X-MAC. An analysis of the traces yielded that retransmissions and the duplicate segment dropping features increased throughput for longer routes, but had a slightly adverse impact for shorter routes. For LPP, the introduction of the caching mechanisms had almost no impact on the end-to-end throughput at all. The curves both exhibit an astonishingly similar degradation of the throughput with increasing route length.

With ContikiMAC, the introduction of the *initial cctrl* module dramatically decreased the amount of transmitted TCP packets. Analyzing the traces, we presume that ContikiMAC suffers more from increased levels of interference and competing

medium access, which is probably triggered by the *early* local retransmissions of the *cctrl* module. The degrading effect of such early retransmissions is higher in ContikiMAC than, e.g., in X-MAC, since ContikiMAC, after knowing the schedule offsets of its neighbors, only transmits the data frames at the announced wake-up time of the targeted receiver. A collision at this point then inevitably results in a transmission failure. In contrast, X-MAC sends out long preamble strobes preceding every frame transmission, where a collision of two strobe packets has no dramatic impact. The preamble strobes further serve to reserve the channel, since they are likely to be overheard by neighboring nodes checking the channel for transmission, probably even by nodes which are more than one hop away in the testbed topology, which alleviates the *hidden node* problem.

Figure 9.14 depicts the radio on-time t_{on} of all the nodes in the chain combined and divided through the number of successfully transmitted and acknowledged TCP segments from the TCP clients (senders) to the TCP servers (receivers). Contiki's internal power profiler [50] calculates t_{on} as the combined duration the radio spends in receive and transmit mode. Since the radio is in general by far the most power-hungry component of a WSN node, the estimation t_{on} has recently often been used for estimating the energy consumption, e.g. in the studies by *Dunkels et al.* [48] and *Boano et al.* [21]. As the scope of the investigation in this chapter is not particularly focusing on energy-efficiency and energy-consumption alone, but rather the achieved end-to-end TCP throughput, we sticked to the radio on-time estimator of Contiki, and neglected to capture physical measurements using SNMD devices (cf. Section 2.3.2 of Chapter 2), since this would have been cumbersome to achieve within the distributed testbed and up to 13 nodes synchronously involved in an ex-

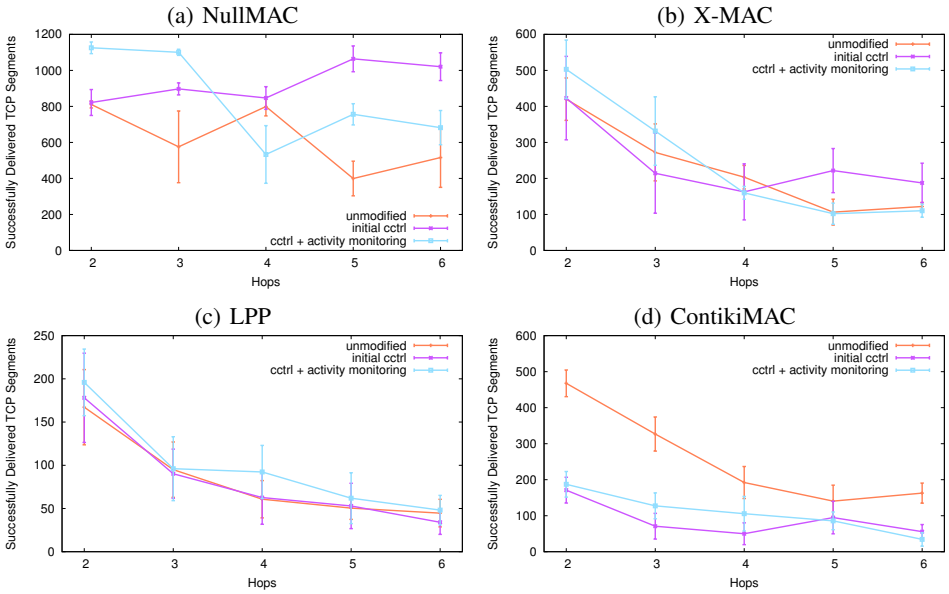


Fig. 9.13: Throughput with *unmodified* Contiki, the initial *cctrl* Strategy *without* and *with* the Activity Monitoring Extension

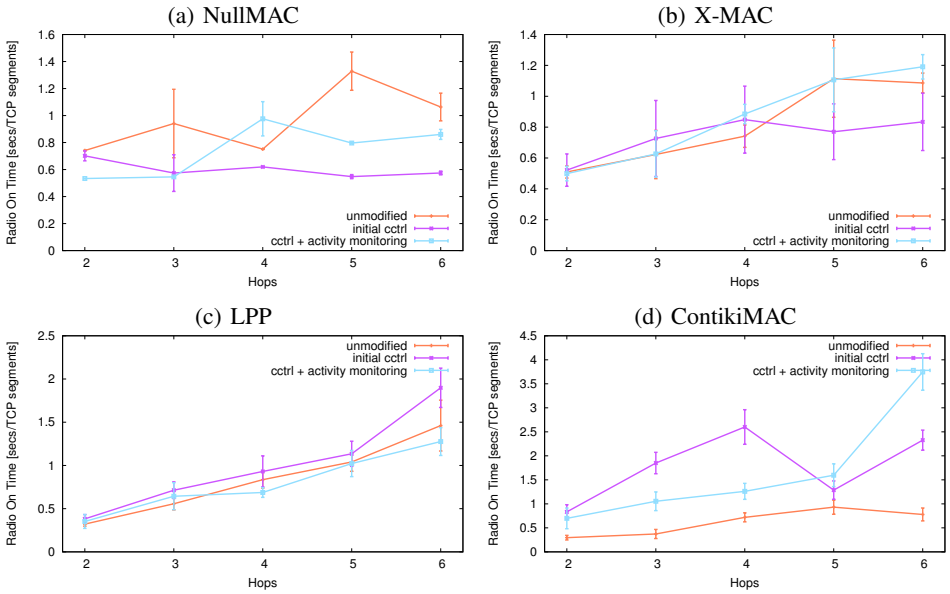


Fig. 9.14: Energy Consumption measured as Radio On-Time t_{on} with *unmodified* Contiki, the *initial cctrl* Strategy without and with the Activity Monitoring Extension

periment. The curves in Figure 9.14 represent a metric for the *energy-efficiency* of the different configurations, since they denote how much radio on-time has been spent per TCP segment on average. The displayed ratio tends to increase with the number of hops for the E^2-MAC protocols, since longer routes obviously require more energy to be spent per segment. In absolute numbers, all protocols operate in a similar range. The energy-unconstrained NullMAC protocol combined with our *cctrl* module even outperforms X-MAC and LPP for most route lengths, since it can transmit much more segments within the 10 min experiment time. The introduction of the *initial cctrl* mechanism tends to improve the energy-efficiency of NullMAC and X-MAC, but clearly deteriorates that of ContikiMAC.

Channel Activity Monitoring

We examined the impact of the channel activity monitoring approach introduced in Section 9.2.3, where the MAC proxy is introduced between the µIP and the MAC layer, in order to make information regarding the current channel utilization available to the *cctrl* module. The strategy then consists in making the *cctrl* module's local retransmission timeouts *dependent* of the registered activity level, transmitting *earlier* in situations of low channel activity (to avoid long wasted idle periods of the channel), and prolonging it when transmissions from neighboring are detected. We refer to this strategy as *activity dependent retransmissions* hereafter, and label it as *cctrl + activity monitoring* in the figures.

Figure 9.13 illustrates how this strategy affects the achieved throughput, and compares it with the *initial cctrl* strategy. NullMAC and X-MAC convey a similar

behavior: for both, the *activity dependent retransmissions* increase the end-to-end TCP throughput for the 2-hop and 3-hop experiments, but decrease it for longer routes, compared to the *initial cctrl* strategy. With X-MAC the throughput even falls below the *unmodified* Contiki configuration for long routes. The LPP throughput could be improved across all route lengths, which resulted in the highest amount of successfully transmitted TCP segments with this MAC protocol so far. When comparing the maximum throughput in absolute values, however, LPP remains far behind ContikiMAC and XMAC, which both operate on a similar level. The energy-unconstrained NullMAC, in contrast, reaches a throughput that is at least 2-3 times as high for all route lengths. ContikiMAC also slightly benefited from the activity monitoring approach. Its throughput was increased for shorter routes (2-4 hops), or remained at roughly the same level for longer routes. However, the throughput of ContikiMAC remained remarkably below that of its configuration without any *cctrl* mechanisms. Obviously, concurrent channel activity and competition is particularly harmful for ContikiMAC, an observation that was later confirmed in the second scenario as well.

The energy-efficiency metric of the *activity monitoring* approach, calculated as radio on-time per TCP segment, is further depicted in Figure 9.14. Compared to the *initial cctrl* strategy, no significant changes could be observed. This result was rather expected, since the MAC proxy remains transparent to the MAC protocol, only passing the gathered channel activity information to the *cctrl* module without introducing any energy costs whatsoever.

Multiple Connections

In Section 9.2.4, we outlined our concept of multiple TCP connection between the TCP client and server, which remains transparent to the application, and through which data can be continuously transmitted. During a disruption in the packet flow of one of the connections, e.g., due to a packet loss, the second connection can still operate, and the channel is not left idle until a retransmission is triggered.

We refer to this approach as *dualconnection* in the subsequent figures of this chapter. First, we evaluated whether initiating two connections without the *cctrl* basic mechanisms increases the end-to-end throughput at all. Thereafter, we examined the initial *cctrl* caching strategy *with* a second connection (cf. *cctrl + dualconnection*) but *without* the activity monitoring, and then in combination with the latter (cf. *cctrl + dualconnection + activity monitoring*). Figure 9.15 illustrates the throughput of these approaches, together with the *unmodified* Contiki and *initial cctrl* strategies examined beforehand.

Again, the different strategies conveyed different results for the four examined MAC layers. With NullMAC, the availability of a second connection and the initial *cctrl* caching and retransmission strategy (cf. *cctrl + dualconnection*) effectively doubled the amount of transmitted TCP segments, across almost all tested routes, compared to the *unmodified* Contiki variant. If, in addition, the *cctrl* mod-

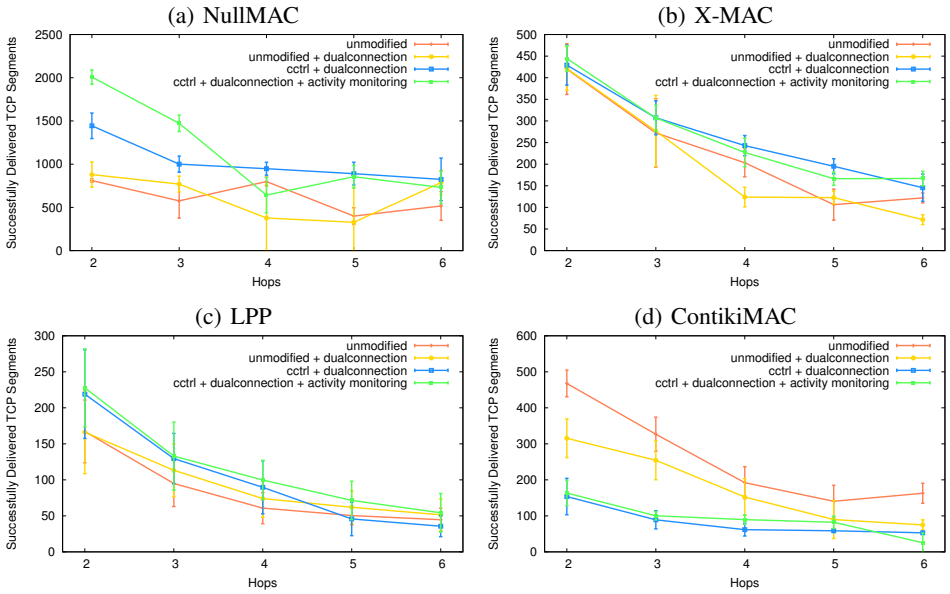


Fig. 9.15: Throughput with Activity Monitoring and Multiple Connections

ule applied the *activity dependent retransmission* strategy, together with the dual-connection approach, the throughput was even further increased for the 2-hop and 3-hop routes, and performed only slightly worse than the *cctrl + dualconnection* approach for longer routes. The X-MAC protocol also benefited from a second open connection, but only if the *cctrl* module was active. Its throughput was increased consistently across all route lengths, in the best case by roughly 37% (cf. 5 hops). Similar results were obtained for the LPP protocol: the experiment configurations relying on two TCP connections significantly increased the throughput, and the best results were obtained by combining the former with activity dependent retransmissions (cf. *cctrl + dualconnection + activity monitoring*).

The ContikiMAC protocol consistently remained the exception in our evaluation with the four examined MAC protocols, and still did not show any signs of TCP throughput improvement. No matter whether ContikiMAC was run with the *cctrl* module combined with a second connection, the activity dependent retransmission strategy or both extensions combined, its throughput persistently remained at a very low level. Again, we presume that the throughput degradation was mainly caused by early triggered retransmissions colliding with the original transmissions or returning TCP acknowledgements, probably because of the *hidden node* problem, which may occur more often with ContikiMAC due to the lack of preamble strobes in advance of frame transmissions. Interestingly, none of the MAC protocols profited from having a second open TCP connection without the *cctrl* module (cf. *unmodified + dualconnection*). Most protocols' throughput remained on a similar level, except for ContikiMAC. With ContikiMAC, this approach achieved a much lower performance, yet at least the overall second-best. The obtained values

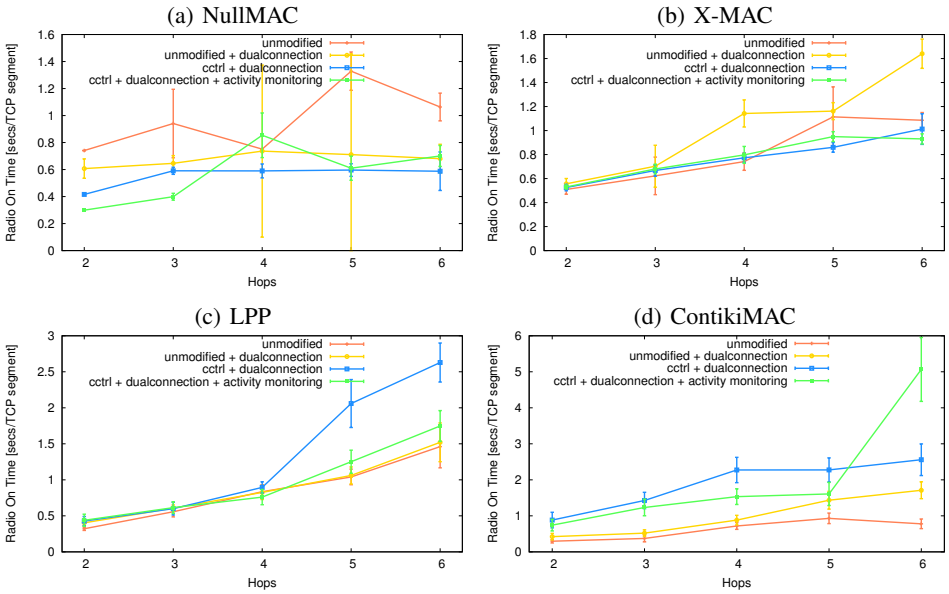


Fig. 9.16: Energy Consumption measured as Radio On-Time t_{on} with *unmodified* Contiki, the initial *cctrl* Strategy without and with the Activity Monitoring Extension

were still consistently below the performance of the *unmodified* Contiki network stack configuration, where only one TCP segment/acknowledgement is in flight.

Figure 9.16 depicts the radio on-time of all the nodes in the chain combined and divided through the number of successfully transmitted and acknowledged TCP segments, which represents an indication for the energy-efficiency of the protocols. One can clearly see the positive impact on NullMAC, where the three configurations *with* our *cctrl* module and the *dualconnection* strategy again achieved a much lower per-segment energy cost. With X-MAC, a slight improvement can be claimed with the strategies *cctrl* + *dualconnection* and *cctrl* + *dualconnection* + *activity monitoring*. With LPP, the efficiently is slightly degraded with long routes, and with ContikiMAC clearly deteriorated for all route lengths.

Overall Comparison

The obtained results provide a detailed insight into the performance of the proposed *cctrl* strategies for the different route lengths. Since the results sometimes vary heavily across the different configurations and route lengths, a general conclusion and guideline, however, remains hard to derive. We therefore averaged the sum of the mean values of each different route length and configuration (= mean value of the five hop-specific mean values per protocol and configuration) to obtain a single value for each examined strategy. Since in WSNs, data is transported from and to nodes at varying distances, averaging the throughput values results in a meaningful number, with which the different approaches become somewhat comparable.

Figure 9.17 displays the obtained averaged values for each examined approach and

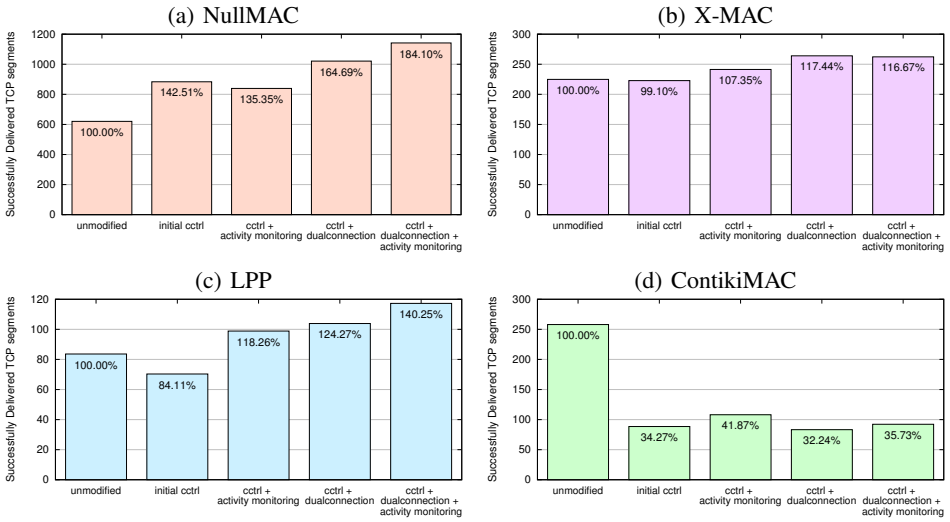


Fig. 9.17: Single Route Scenario: Comparison of Throughput

each MAC protocol. The application of distributed caching and local retransmission of TCP segments and acknowledgements, denoted as *initial cctrl* approach, obviously already had an astonishingly good impact on the NullMAC protocol. Across all route lengths, this approach reached an improvement of 42.51% compared to *unmodified* Contiki. Our proposed extensions of *activity dependent retransmissions* as well as the *multiple connections* combined finally reached the best results, with an average increase of 84% compared to the unmodified Contiki µIP configuration. With the X-MAC protocol, the improvement is less distinctive. When following the *initial cctrl* strategy, the performance remained almost equal to the unmodified Contiki variant. The best results of 17% more transmitted TCP segments could be achieved when combining the initial *cctrl* approach with a second TCP connection. Similar results were obtained by combining this strategy with the activity monitoring. LPP behaved similarly as X-MAC when applying the different *cctrl* extensions. Again, our modifications of *activity dependent retransmissions* and the *multiple connections* combined achieved the best overall performance with X-MAC, an end-to-end throughput increase of remarkable 40%.

Figure 9.18 depicts the energy-efficiency metric calculated as radio on-time per transmitted TCP segment for the four protocols. The efficiency of NullMAC clearly profits from the *cctrl* module, in particular when combining all the strategies. Since NullMAC reaches the highest number of transmitted TCP segments over the entire experiment timespan, its efficiency is even better than that of any E^2-MAC protocol (< 0.5 seconds per TCP segment). The energy-efficiency of X-MAC is slightly improved with most of the mechanisms of the *cctrl* module. LPP remains in the same range as X-MAC in absolute values. However, its efficiency varies rather strongly with the different strategies, but is neither deteriorated nor improved. ContikiMAC again constituted the negative exception among the four evaluated

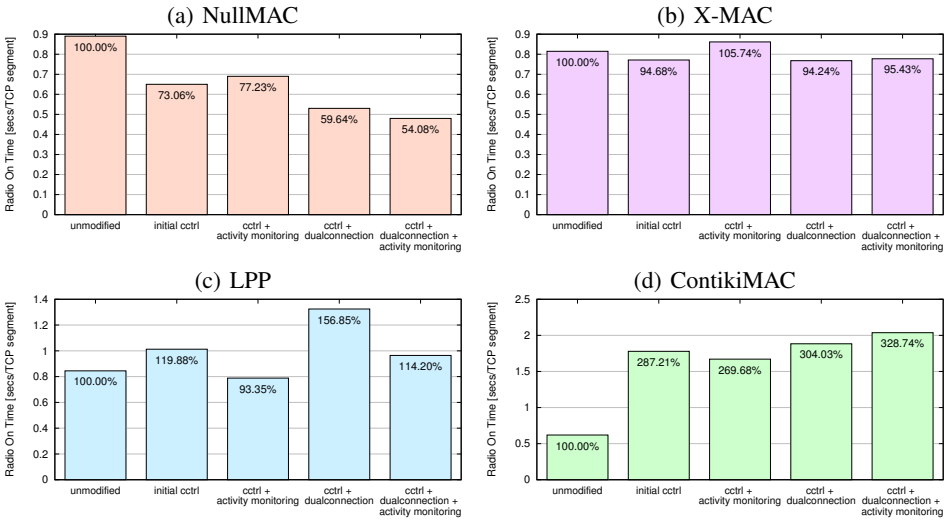


Fig. 9.18: Single Route Scenario: Comparison of Energy Consumption per TCP Segment

protocols: its energy-efficiency suffers a lot when introducing any caching strategies whatsoever. This degradation can most probably be explained by the increase in collisions and other sources of packet losses due to concurrent activity in the channel, which does not occur when only one TCP segment is in flight at any time and no local retransmissions are triggered, which is only the case with the *unmodified* variant. Since ContikiMAC acknowledges the data packets themselves, and re-attempts to transmit upon failed attempts, as opposed to X-MAC, it already integrates certain reliability. Comparing the similar, yet even slightly better results of the X-MAC configuration with all our extensions (*cctrl* + *dualconnection* + *activity monitoring*) with that of unmodified ContikiMAC, the question whether reliability should rather be ensured on a *hop-by-hop* or *end-to-end* manner can not be answered conclusively.

9.3.3 Cross Traffic Scenario

With the *Cross Traffic Scenario*, we aimed at scrutinizing our findings in an environment where the route along which TCP operates is prone to intensive interference. The topology of this scenario is illustrated in Figure 9.12. Two routes span across three floors, and share one common node (node 3) in the top floor. Furthermore, the experiment settings are the same as in the single route case: each route contains a TCP client that tries to send as many TCP data segments to the TCP server node placed on the other end of the route. Node 3 being part of both routes has to forward traffic from both TCP connections. Again, we tested the network topology with different lengths of the routes.

The majority of the examined MAC layer protocols conveyed the best or nearly the best results in the case of a combined application of the proposed features. Since we were mainly interested in verifying that this configuration achieved the best per-

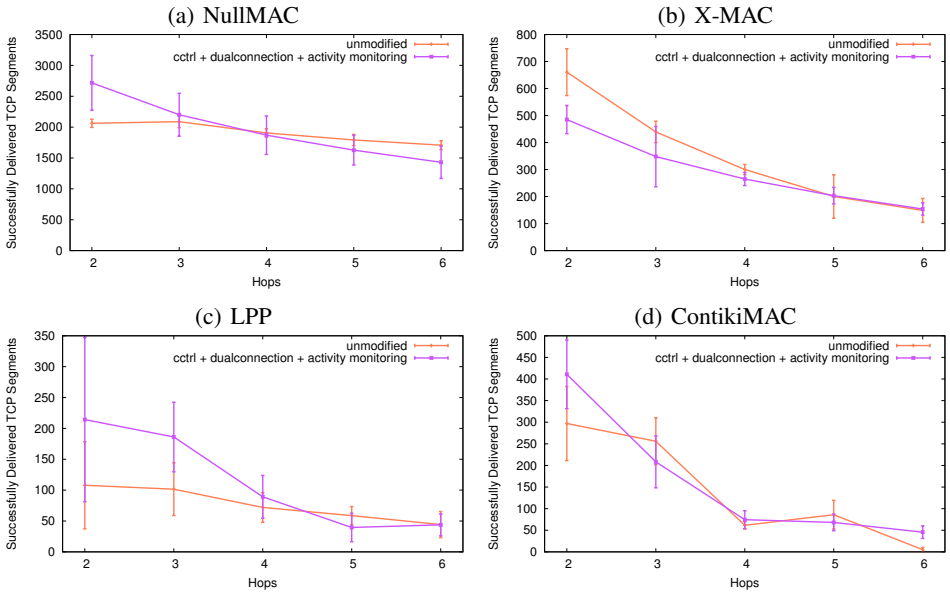


Fig. 9.19: Throughput with Activity Monitoring and Multiple Connections

formance, we decided to limit the evaluation with the cross traffic scenario to the *cctrl* module only to this particular configuration. The configuration that showed the best overall performance in the previous single route experiment is that of *activity dependent local retransmissions* combined with a second TCP connection, i.e., the rightmost bar in Figure 9.17. We will refer to this configuration again as *cctrl + dualconnection + activity monitoring* in the subsequent figures. Figure 9.19 compares the total amount of successfully received TCP packets over both established routes at the nodes 1 and 2 in Figure 9.12. Figure 9.20 again illustrates the same results averaged across all path lengths.

For all tested protocols, the throughput decreases gradually with the hop distance, a behavior that has been observed many times. The degradation of throughput across n hops approximately follows the $\frac{1}{n}$ rule, where n equals the number of hops the data has to travel, as outlined by *Österlind et al* [137]. Besides this well-investigated insight, the results obtained in the *Cross Traffic Scenario* turned out to differ quite heavily from those obtained in the prior *Single Route Scenario*. NullMAC again achieved the highest throughput of all MAC protocols, with an end-to-end throughput around 2000 segments across 6 hops in the unmodified Contiki variant. The *cctrl* module's caching and local retransmission mechanism, coupled with the *activity monitoring* and the *multiple connections* strategy, which had reached the best results in the single route experiment, increases the throughput for NullMAC in the 2-hop and 3-hop case. However, it performed slightly worse with longer routes. The massive improvement of 84% obtained with the *cctrl + dualconnection + activity monitoring* configuration in the *Single Route Scenario* (compared with the unmodified Contiki µIP stack, cf. Figure 9.17) could not be reproduced in the *Cross Traffic Scenario*.

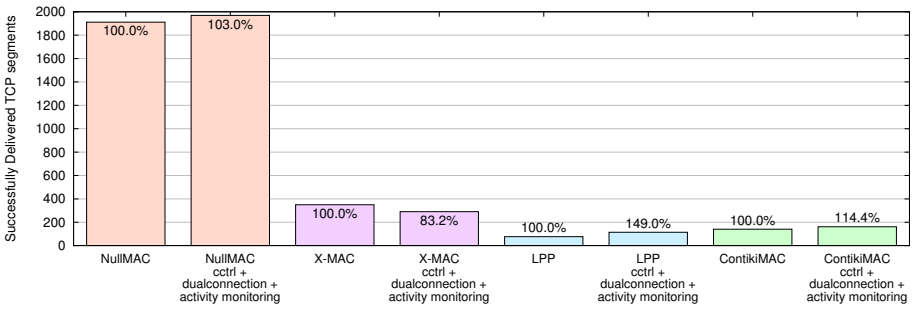


Fig. 9.20: Cross Traffic Scenario: Comparison of Throughput

With X-MAC, the slight improvement of the TCP throughput observed in the *Single Route Experiment* could also not be reached. In fact, the protocol performed slightly worse for short routes, and equally for longer routes. X-MAC obviously did not cope well with the additional interference caused by the local retransmissions performed by the *cctrl* module in an environment with already increased traffic load, as revealed with the 2-hop and 3-hop results. Yet, X-MAC performed significantly better than its successor ContikiMAC in the *Cross Traffic Scenario*, no matter whether the *cctrl* module was activated or not. The LPP protocol exhibited the highest improvement induced by the *cctrl* module and the extensions. Throughput could be improved particularly for short routes (2-hops and 3-hops), where it almost doubled. For 4, 5 and 6 hops, LPP's throughput decreased to the level of that of the unmodified variant. We assume that the increasing number of nodes, which periodically transmit beacons to indicate reception readiness, increasingly led to collisions. However, our approach managed to improve the TCP throughput by 49% on average, which is an even larger improvement than that of LPP in the *Single Route Scenario*.

When comparing the absolute numbers of transmitted TCP segments of the *Cross Traffic Scenario* in Figure 9.19 with those of the *Single Route Scenario* in Figure 9.17), one can clearly see that with NullMAC, the aggregated throughput of the two routes experiment reaches roughly two times that of the single route experiment. With X-MAC and LPP, the aggregated throughput of the two routes is in the range of 1.5 times that of one route. However, with ContikiMAC, the average amount of successfully delivered TCP segments of the *Cross Traffic Scenario* drops to less than half of that obtained in the *Single Route Scenario*. This observation was made across all route lengths. At the 6-hop run, the end-to-end throughput almost stalled completely. Hence, we have again observed that ContikiMAC has yet major difficulties in environments with increased levels of interference and nodes concurrently competing for channel access. The medium reservation and contention mechanism of ContikiMAC should probably be verified under increased network contention in a controlled environment. NullMAC, in contrast, seems to exhibit no major difficulties with concurrent traffic within the building. Interferences from transmissions on the other route generally deteriorated the throughput of the E^2-MAC protocols more than that of the energy-unconstrained NullMAC.

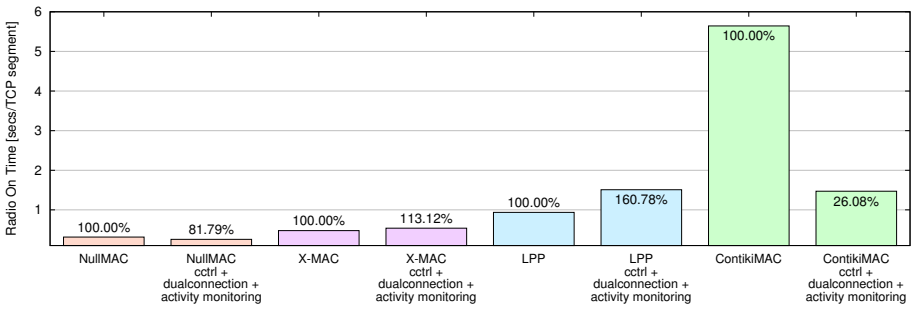


Fig. 9.21: Cross Traffic Scenario: Comparison of Energy Consumption per TCP Segment

The introduction of the *cctrl* variant in the *Cross Traffic Scenario*, which had resulted in a throughput decrease in the *Single Route Scenario* with ContikiMAC, now lead to a slight *increase* of its throughput. On average, an improvement of 14% could be achieved. Based on an analysis of several trace files, we conjecture that this observation can most probably be explained as follows: ContikiMAC suffers from interferences with the second route, which often leads to packet losses. With long routes, chances are high that the two active routes at some point interfere with each other and cause a packet loss, which explains the significant drop of ContikiMAC at 6 hops, where not a single TCP segment made it through the chain in all 15 experiment runs. Without the *cctrl* module, each packet loss leads to an end-to-end retransmission, however, again with the same low success probability. With the *cctrl* module, a packet lost in the middle has a higher probability not to collide until it reaches its destination. Therefore, the 6-hop throughput of ContikiMAC *with* the *cctrl* module is rather low, but at least remains above zero.

Figure 9.21 depicts the impact of the examined *cctrl* variant on the energy consumption, again measured as radio on-time per TCP segment, which represents a metric for the energy-efficiency. Furthermore, the figure puts the E^2-MAC protocols' performances into relation. Since NullMAC reaches the highest number of transmitted TCP segments over the entire experiment timespan, its per-segment value tops the performance of each of the radio duty-cycling MAC protocols. The introduction of the *cctrl* module even improves the efficiency by another 18%. The protocols NullMAC and LPP consistently profited from the application of the *cctrl* module, although for NullMAC, the improvement was rather small compared to the result of the prior scenario. Much to our surprise, ContikiMAC also benefited from using the *cctrl* module, with respect to throughput and energy-efficiency. The average radio on-time per TCP segment was lowered by almost 73%, resulting in the most significant change experienced in the entirety of the conducted experiments. Elaborate analysis of the traces revealed that without the *cctrl* module, ContikiMAC experienced numerous subsequent packet losses, which caused the TCP connection to finally time out. The costly reestablishment of the TCP connection then worsened to ratio between radio on-time and received TCP segments, as no data could be transmitted during this time.

9.4 Conclusions

In this chapter, we have outlined and evaluated our *cctrl* module, a modular add-on for Contiki's µIP stack, which implements and augments the distributed caching and local retransmission features proposed in DTC [46] and TSS [24] within the Contiki OS network stack. On top of the basic retransmission mechanism, we designed and implemented additional extensions for our *cctrl* module. By involving three radio duty-cycling E^2-MAC protocols, we further examined the impact of different MAC protocol approaches to the mechanisms proposed in [46][24], which turned out to be significant.

We tested our implementations in an indoor wireless sensor node testbed in two scenarios where data has to be transmitted reliably across routes of increasing length. In general, we encountered a rather high variability among the results of the different examined protocol configurations. Some protocols (e.g., NullMAC, LPP) generally reacted positively to the local retransmission scheme, whereas others (i.e., ContikiMAC) performed rather worse. The results revealed that the *cctrl* module can definitely increase the throughput of TCP across multi-hop WSNs in many of the examined situations. However, due to the differences in the examined MAC protocols algorithms, the impact of the proposed *cctrl* extensions varied heavily, such that was impossible to find a single *cctrl* configuration that maximizes the throughput with all evaluated MAC layers. We observed that the results depend not only on MAC protocol characteristics, but also on the network topology and the presence of interfering traffic. The increase in throughput ranged from 3% in the *Cross Traffic Scenario* with NullMAC, to up to 84% in the *Single Route Scenario* with the same protocol.

Among the radio duty-cycling MAC protocols, X-MAC in combination with the *cctrl* module achieved the highest average throughput in the single route experiment. In this configuration, it exhibited the most consistent and predictable behavior of all E^2-MAC protocols, taking the results from both scenarios into account. The comparison of NullMAC with the three E^2-MAC protocols in Figure 9.20 furthermore underlines our findings from Chapters 6 and 7: the energy-unconstrained CSMA variant NullMAC reaches by far the highest throughput of all examined protocols. In the *Cross Traffic Scenario*, X-MAC reached less than 20% of NullMAC's throughput, LPP and ContikiMAC even far less than 10%. Measuring the radio on-time per transmitted TCP segment as an indication for the energy-efficiency, NullMAC even outperformed all E^2-MAC protocols.

The basic idea and approach of MaxMAC, which effectively consists in keeping all nodes in a multi-hop chain awake and operating in the CSMA state in case a large portion of data has to be transmitted, is hence justified by the observation that NullMAC reached the best radio on-time per transmitted TCP segments in both experiments of this chapter. According to our observations of Chapter 7, we expect that the application of MaxMAC to the examined experiments of this chapter would convey similar results as NullMAC, with respect to both throughput

and energy-efficiency. In case of an experiment scenario with traffic load varying between peaks of high intensity, and phases of low or even no network activity, MaxMAC could even outperform all the examined E^2-MAC protocols. An implementation of the MaxMAC protocol for Contiki would therefore constitute a significant added value for a wide range of TCP-based applications, which rely on the provision of high throughput and short latencies for brief periods of increased network activity, e.g., for letting the network operator communicate with single nodes for configuration or code updates, and standard TCP applications such as Telnet [148], SMTP [145] or FTP [146].

Chapter 10

Conclusions and Outlook

Whereas wireless sensor network technologies have just attracted the attention of the academic community one decade ago, a large number of real-world use cases and business models have emerged in the meanwhile. Yet, wireless sensor network technologies are far from having exploited their full potential. Researchers and developers working in the WSN field still face a large number of limitations and problems, which are yet to be solved. In this thesis, we have evaluated the feasibility and potential of run-time adaptive resource allocation schemes in several contexts, thereby addressing four main challenges and problem categories. Section 10.1 describes the four main challenges we tackle with the contributions of this thesis. These contributions are summarized in Section 10.2. Section 10.3 then outlines promising research topics for future work related to our contributions.

10.1 Addressed Challenges of the Thesis

These addressed challenges and problem domains, to which we presented a number of contributions in the different chapters of this thesis, address the following four problem categories and challenges in WSN research:

- **Experiment Methodologies:** The usual approach many researchers follow today consists in rapid-prototyping a WSN algorithm or application first by using a network simulator, testing and comparing it to existing approaches, and proceeding with implementing a real-world prototype when the simulation results are promising. *Real-world prototype* based research in experimental testbeds, however, is a tedious and time-consuming task. Many WSN testbeds require physical presence at site and offer only limited run-time information. Erroneous protocol behavior can hence often only be spotted in offline trace analysis. Furthermore, the inconsistent representation of experimental data in today's WSN communities severely aggravates experimental research: often, results from large-scale experimental WSN studies are arbitrarily organized and formatted, which renders reproducibility impossible. Further problems concern the availability of simple and painless experiment methodologies, e.g., to

assess the energy consumption or environmental interferences.

- **Hardware & Software Restrictions:** Forced by the rules of economics, WSN nodes have to be cheap and expendable in order to qualify for a wide range of today's and future WSN applications. Hence, they are usually built from inexpensive electronic circuitry and components. The restrictions with respect to memory, processing power and bandwidth pose inherent challenges to the development of sensor network software, ranging from the application layer to the operating system and the communication stack. In practice, this limitation often means that mechanisms designed and evaluated in a network simulator can not be realized exactly as planned on real-world sensor nodes, but have to be *downsized* and *simplified* in order to meet the hardware restrictions.
- **Energy Restriction:** Most of today's platforms drain out of energy after not much more than a couple of days if all onboard components (sensors, CPU and radio) are permanently kept on. The development of radio duty-cycling MAC protocols has massively increased WSN lifetimes, and has thus been a cornerstone in WSN research. The development of energy harvesting/scavenging techniques based on, e.g., thermal-electric, photo-electric or piezo-electric technologies has yet come up with a range of solutions to alleviate the energy restrictions. However, *Kompis et al.* [102] have outlined that these mechanisms have yet not eliminated the need for frugal use of the energy resources.
- **Wireless Channel Properties:** The low-power wireless channel used in WSNs is inherently unreliable: it is prone to the effects of multipath propagation, reflection, scattering, or interferences with nearby nodes, or devices operating in the same band. Often, links in WSNs are intermittent: communication across one link may succeed during one hour and fail in the next hour, e.g., because of changes in the environment, node mobility, or random node failures. Since transmitting with higher power is energetically expensive, WSNs will also in the future have to cope with low signal-to-noise ratios and the aforementioned typical wireless phenomena. Intelligent and adaptive mechanisms remain to be designed in order to fully cope with the dynamicity of the unreliable low-power wireless channel and the performance-degrading effects (with respect to packet delivery rates, reliability, latency) of typical wireless phenomena.

10.2 Thesis Summary

In the contributions presented in this thesis, we have faced the challenges listed above, and we have addressed them with the frameworks of solutions, protocols and concepts presented in Chapters 3-9. While Chapters 3 and 4 (Part I) describe frameworks and tools for experimental WSN research and evaluations, the subsequent Chapters 5-9 (Part II) present several solutions for run-time adaptive communication protocol architectures in WSNs, which forms the superordinate topic spanning across the different contributions of this thesis.

Part I - Frameworks and Tools

In Chapter 3, we have presented the Testbed Management Architecture for Wireless Sensor Network Testbeds (TARWIS), our solution for efficient, automated and repeatable experimentation with WSN testbeds. TARWIS remains independent from the physical setup of the testbed structure, the sensor node hardware or software (operating system). With TARWIS, we can offer interested research groups a solution for setting up own testbeds, and relieve them from the burden to implement own user administration, experiment scheduling and testbed management solutions from scratch. TARWIS to date runs on nine different testbeds of wireless sensor networks of the European-Union WISEBED [162] project, with node deployments between a few 10 to more than 100 nodes. Throughout the course of the WISEBED [162] project, we have set up a testbed of 47 sensor nodes and have made it available to researchers from all over the globe. We have further conducted the major part of experiments described in this thesis using TARWIS, among them most of the results of Chapters 7, 8 and all the results of Chapter 9. The major advantage of TARWIS is the simplicity and convenience with which a real-world sensor network testbed can be operated. Its lucid user interface permits to observe and inspect protocol behavior in a high granularity at experiment run-time, instead of tedious offline trace analysis. Without exaggerating, one can well claim that the ease of using TARWIS reminds that of many of today's network simulation tools.

In Chapter 4, we have discussed different *software-based estimation* methodologies to assess the energy consumption of WSN nodes. Software-based energy estimation yields significant advantages over energy measurement using digital storage oscilloscopes or high-resolution multimeters: only software-based *on-line* energy estimation mechanisms running on the node themselves enable the network to take energy-aware decisions about routing, clustering or transmission power scheduling. We have contributed to this research domain with a profound empirical evaluation of the accuracy of different software-based estimation models and parameter calibration techniques. Chapter 4 identifies and quantifies the different factors that cause deviations of the software-based estimations from the *real* physically measured energy consumption. The inaccuracies in the production of the electronic components have been shown to impact on different power consumption levels, which led to nodes differing by more than 4% in their average energy consumption. The most widely used software-based energy model, referred-to as *Three States Model* in Chapter 4, has been shown to result in an estimation error ($\mu \pm \sigma$) of 3.00 % $\pm$ 2.55%, which yet suffices for a large range of experimental studies, but which yields potential for improvement. Finally, we have shown that our proposed methodology of enhancing the estimation model with information regarding the state transitions, and applying multivariate OLS regression to calibrate the model parameters using empirical data, can remarkably reduce the resulting estimation error. With our protocol-independent methodology, we measured a mean absolute estimation error ($\mu \pm \sigma$) of 1.13% $\pm$ 1.15%, an accuracy that clearly suffices for the vast majority of experimental studies on WSN protocols.

Part II - Contributions To Communication Protocols

In Part II, we proposed and evaluated a range of mechanisms and protocols that apply run-time adaptive resource allocation within different layers of the ISO/OSI communication protocol stack. The contributions relate to each other with respect to their simple state-based approach of allocating more resources (e.g. radio on-time, computational power) when crucial variables of the environments change during run-time (e.g., threshold parameters exceeded, errors reported in ACKs), and where hence the prerequisites are given to apply countermeasures to prevent from a degradation of service quality.

We commenced our contributions with the evaluation of the WiseMAC burst transfer mode *More Bit* and our proposed *Extended More Bit*, both in simulation and on a real-world sensor node platform. Our results confirmed that the *Extended More Bit* basing on the receiver *promising* to remain awake after a packet burst performs better than the original WiseMAC *More Bit* scheme with respect to the achievable throughput. The superior performance of 20%-25% has been found similar in both simulation and real-world experiments. This study published in [87][91][89] ignited our interest on throughput-maximizing but at the same time energy-efficient mechanisms on the MAC layer, which finally led to the more sophisticated traffic adaptivity mechanisms designed, implemented and evaluated in Chapter 7.

In Chapter 6, we explored the current state of the art in E^2-MAC protocol design, in particular by examining today's most frequently cited E^2-MAC protocols with respect to their ability to react to variable traffic conditions. By comparing against an idealized concept of an E^2-MAC protocol named IdealMAC, we have shown how far today's E^2-MAC protocols still are from the goal of being able to truly allocate the radio transceiver in an on-demand manner. The evaluation clearly shows by means of network simulation that mainly all existing E^2-MAC protocol heavily restrain the maximum achievable throughput for the upper layers, and do not provide adequate measures to adapt to variable load. We have developed a tripartite metric to measure and quantify the *traffic adaptivity* of an E^2-MAC protocol by taking into account the crucial target variables maximum achievable throughput, latency and energy-efficiency. With so-called traffic adaptivity (TA) metric, we have formalized a notion for the property of *traffic adaptivity*, which so far has been referred to often in an ambiguous and indeterminate manner. Applying the TA metric to the selection of MAC protocols conveyed that the WiseMAC [57] yet achieves the best TA values under variable load, a conjecture that is also supported by the recent comparative study of *Langendoen* [107].

The Maximally Traffic-Adaptive MAC (MaxMAC) protocol introduced in Chapter 7 is our contribution proposed to fill the gap of run-time traffic-adaptive MAC protocols discovered in the *state of the art* analysis beforehand. The protocol integrates basic design principles of preamble-sampling based random access MAC protocols, in particular WiseMAC and X-MAC. Based on a simple finite-state-machine-based adaptation model, the protocol is able to allocate more energy-resources when more load has to be handled, or even completely abandons any

radio duty-cycling sleep-wake pattern when necessary. We evaluated the protocol's behavior in a series of experiments in a network simulator, and compared it against our idealized concept of an E^2-MAC protocol using the TA metric defined in Chapter 6. We then studied the real-world feasibility of the MaxMAC protocol with our MaxMAC prototype implementation and compared its behavior against two other wireless MAC protocols. Both experimental evaluations demonstrated that MaxMAC reaches its goal of being clearly distinguishable from existing preamble-sampling based approaches by reaching nearly the same throughput and a similarly low latency as energy-unconstrained CSMA, while still exhibiting the same energy-efficiency during periods of sparse network activity. The MaxMAC protocol hence succeeds in combining the advantages of energy unconstrained CSMA (high throughput, high PDR, low latency) with those of classical E^2-MAC protocols (high energy-efficiency). Like most contention-based MAC protocols, MaxMAC is a general-purpose protocol, and does not rely on assumptions which are cumbersome to achieve (e.g, rigid time-synchronization across the entire network). It can hence be applied without changes in scenarios where constant low-rate traffic is expected, and where in most cases B-MAC and X-MAC are being used today.

Chapter 8 addresses the challenges related to the inherently unreliable wireless channel in WSNs. The chapter explores the potential of Forward Error Correction (FEC) schemes. We have implemented eight different Error Correcting Codes (ECCs) in our library *libECC* and have proposed three run-time adaptive FEC strategies, which react to deteriorating link quality by allocating the correctional power of ECC codes in an *on demand* manner. The concept applied in this context shares many similarities to that of Chapter 8. The parameter adaptation algorithm follows a similar finite-state-machine-based model, where each state describes a certain set of parameters. Input variables such as the success of previous transmissions then define the state transitions - allocating less computational power when the link quality is good and unencoded transmissions are successfully acknowledged, and more sophisticated coding when link qualities are deteriorated. We have examined our different implemented ECCs as well as the adaptive FEC strategies in a series of experiments, of which the majority were conducted again using TARWIS. The main conclusions drawn from Chapter 8 is that the packet delivery rate (PDR) could generally be increased on weak and lossy links using FEC mechanisms, and that the occurrence pattern of transmission errors turned out to be a rather *local phenomenon* that differs heavily from link to link. Consequently, the proposed adaptive FEC schemes offer significant advantages with respect to conservation of precious CPU time and energy, since they adapt the correctional power on a per-link basis, deciding *when* and *where* to apply FEC in a totally distributed manner. This turned out to significantly impact on the achieved end-to-end PDR at a reasonable and limited overall energy cost.

In our final contribution presented with Chapter 9, we experimentally evaluate the performance of TCP/IP across multiple hops in WSNs. TCP/IP has been shown to

perform rather poorly [46][24] in WSNs with multiple hops, due to the unreliable nature of the wireless channel (higher bit error rates and packet loss), particular properties of and interactions with the underlying wireless channel MAC protocols (exponential backoff mechanisms, hidden node and exposed node problem), and the design of the TCP congestion control mechanisms. We take up basic concepts of distributed TCP caching and local retransmissions proposed in [46][24] and self-developed extensions of the latter, and implement them - in contrast to the studies [46][24] themselves - in a MAC-layer independent manner into our *Caching and Congestion Control (cctrl)* module. The *cctrl* module developed within this experimental study is a modular add-on for the µIP stack [45] of the Contiki OS [47]. We show that our contribution is able to significantly increase the end-to-end throughput across multiple hops in various real-world WSN topologies. We study the performance of *cctrl* using three different E^2-MAC protocols (X-MAC, LPP, ContikiMAC) and Contiki's energy-unconstrained CSMA variant NullMAC. The TARWIS testbed management architecture, as discussed in Chapter 3, was used to schedule and execute all the experiments of this chapter. The main results of this study can be summarized as follows: our *cctrl* module managed to increase the end-to-end TCP throughput of NullMAC by up to 84% across routes of 2-6 hops lengths. The E^2-MAC protocols X-MAC and LPP tended to exhibit improvements, whereas ContikiMAC's performance was significantly degraded. Much to our surprise, the best results with respect to the energy-efficiency (measured as radio on-time per TCP segment) were achieved with NullMAC in combination with our *cctrl* module. If a large portion of data has to be transmitted to a certain node across multiple hops, it is a better strategy to let the entire route temporarily operate without duty-cycling the radio, because the net radio on-time per transmitted TCP segment is lower than with any existing E^2-MAC protocol. This observation basically confirms and justifies the MaxMAC concept of Chapter 7, which proposes to temporarily switch to CSMA if the encountered load conditions can not be handled anymore without major packet loss when sticking to the periodic radio duty-cycling pattern.

10.3 Thesis Outlook

The work conducted in this thesis opens a broad range of possible directions for future research and innovation. We briefly elaborate on some of the most promising topics that relate to our contributions.

The Testbed Management Architecture TARWIS presented in Chapter 3 definitely made many experimental evaluations of this thesis less cumbersome and error-prone, since it fully automates the experimentation process and requires no continuous physical presence at the testbed site. However, with yet only capturing the raw serial output of the WSN nodes in the testbed, many aspects of WSN experimentation are still left aside. Node failures can yet only be detected, but pinpointing the actual reason behind them still often remains impossible. TARWIS could be

augmented to sample the energy-consumption, the received power level, the temperature or pressure around each installed node, or other variables, which potentially impact on its run-time behavior. TARWIS and the WiseML standard are only first steps towards convenient, repeatable experimentation and a consistent notion of experiment results. An integration of the TARWIS testbed management services with an Integrated Development Environment, which would permit researchers to conveniently upload and test their sensor node code from within their workbench, would be another visionary perspective for TARWIS.

The software-based energy-estimation methodology discussed in Chapter 4 basing on a fine-grained estimation model should be integrated into the more popular sensor node operating systems Contiki OS [47] and TinyOS [112], in order to make this key contribution of the thesis available to a broader research community than the ScatterWeb^2 OS [157] users alone, which, as a matter of fact, is very small. The proposed model extensions that consist in taking into account transceiver switches as regressors into a multivariate OLS model, could further be applied to model the onboard physical sensors. Other components, which are frequently encountered on sensor node platforms, e.g., EEPROM or SD memory slots, could likewise be integrated into such a model. A preliminary study on the integration of further sensor node components has been conducted in [136], where besides the radio, the temperature and humidity sensor are similarly modeled and the respective parameters calibrated. The results with respect to the obtained measurement accuracy basically confirms the adequateness and generality of the multivariate OLS-based approach. The empirical determination of OLS model parameters and the resulting estimation accuracy for a range of WSN platforms could form the topic for several semester or bachelor thesis projects. First steps to integrate our contributions of this chapter into the Contiki OS power profiler have been undertaken recently.

The MaxMAC protocol in Chapter 7 has been evaluated in a lab and testbed environment with rather artificial traffic patterns, except for the use case oriented scenario in Section 7.4.4. In order to fully evaluate the advantages and drawbacks of MaxMAC, a long-term real-world deployment in an event-based application (e.g., a study related to environmental monitoring [183] or a health-care related application [119]), during which essential long-term operations would have to be applied (e.g., code and configuration updates) would outline advantages and drawbacks, but also further potential for improvements. An implementation on the popular sensor node operating systems Contiki OS [47] and TinyOS [112] would further increase the range of potential users. Since the WSN field has clearly shifted towards platforms equipped with IEEE 802.15.4-compliant chips in the past couple of years, a platform with a packetizing radio should be envisaged for this purpose. An interesting research question would furthermore consist in designing an efficient distributed algorithm to define the number of intermediate states and their threshold parameter values in MaxMAC, e.g., based on the knowledge of the network size and topology, probably by specifying the network topology and the requirements with respect to mean latency and throughput. Heuristics based on these

parameters would significantly improve the general applicability of MaxMAC on other real-world platforms. As already discussed in Chapter 7, a distributed parameter learning algorithm, which attempts to meet predefined Quality of Service target goals specified by the network operator, would be the favorable and most elegant solution. Such an algorithm should translate end-to-end Quality of Service goals to individual parameter sets for each node within the network, taking the discovered parameters (i.e., transceiver bandwidth, energy consumption, network topology, etc) into account, and would form a promising research topic for future investigations.

The Forward Error Correction schemes discussed in Chapter 8 have been evaluated using a byte-level radio transceiver operating in the 804-940 MHz ISM frequency band, using simple on-off keying (OOK) as modulation scheme. As mentioned beforehand, the WSN research field and the related industry has clearly moved towards standardized and IEEE 802.15.4-compliant radio chips in the 2.4 GHz ISM band, and the so-called Offset Quadrature Phase-shift Keying (OQPSK) modulation scheme. *Liang et al.* [114] have examined one Hamming and one Reed-Solomon variant to protect transmissions of a TelosB network from interferences with an IEEE 802.11b/g wireless LAN, which typically introduce large burst errors. To the best of our knowledge, a study examining the potential of (adaptive) FEC schemes to cope with low signal-to-noise ratios, and high bit error rates due to multipath propagation, reflection or scattering effects carried out in the 2.4 GHz band does not yet exist. An evaluation of our DECTED, the five BCH variants, and the simple bit repetition codes of *libECC*, as well as the adaptive FEC strategies, would most probably convey different results on this kind of radios, since bit error patterns in wireless communications often heavily depend on the modulation scheme and the radio frequency, and would form an appealing topic for future investigations.

The *Caching and Congestion Control (cctrl)* layer presented in Chapter 9 has been evaluated on TelosB nodes in our indoor distributed testbed laboratories. The resulting *cctrl* module was designed to remain as independent as possible from a) its underlying MAC protocol and b) from any changes on the µIP stack [45]. In our *activity monitoring* approach and the *activity-dependent retransmission strategy* outlined in Chapter 9, we basically fed MAC-layer parameters to the *cctrl* layer, yet satisfying the MAC-layer independence constraints. Softening these constraints would yield potential for further improvements with respect to the end-to-end throughput - the MAC Proxy could not only count the overheard transmissions, but also inspect the TCP headers within the packets and, depending on the utilized MAC layer, feed this information to the *cctrl* module. With this knowledge, implicit acknowledgements could actually be implemented without making continuous overhearing necessary. In case *cctrl* module would overhear the next node in line forwarding the previously sent TCP segment, it could take advantage of this knowledge and cancel its retransmission timeout for the same segment, knowing that its previous transmission definitely made it to the next node.

Similarly, further potential for improvements could potentially be found by making slight changes on the µIP stack. Recently proposed *bulk-transfer* protocols such as PIP [152] or *Lossy Links, Low Power, High Throughput* [52] allow the transmission of large bursts of TCP numerous segments synchronously *in flight*, and assume that each node can buffer significant amounts of data ($\sim$ 1 MB) on external flash memory. Combining the *cctrl* mechanisms with these bulk-transfer protocols would require modifications on the µIP stack, but would open further room for significant improvements.

Bibliography

[1] J.-S. Ahn and J. Heidemann, "An Adaptive FEC Algorithm for Mobile Wireless Networks." Technical Report ISI-TR-555, University of Southern California - Information Sciences Institute, California, USA, March 2002.

[2] J.-S. Ahn, S.-W. Hong, and J. Heidemann, "An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks," *IEEE Journal of Communications and Networks*, vol. 7, no. 4, pp. 489–499, December 2005.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *IEEE Communications Magazine*, vol. 38, no. 4, pp. 393–422, March 2002.

[4] I. Akyildiz, T. Melodia, and K. Chowdhury, "Wireless Multimedia Sensor Networks: A Survey," *Elsevier Computer Networks*, vol. 51, no. 4, pp. 921–960, December 2007.

[5] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Römer, K. Langendoen, J. Polastre, and Z. A. Uzmi, "Medium Access Control Issues in Sensor Networks," *ACM SIGCOMM Computer Communication Review*, vol. 36, pp. 33–36, April 2006.

[6] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," Request for Comments RFC 2581, Internet Engineering Task Force (IETF), April 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[7] I. T. Almalkawi, M. Guerrero Zapata, J. N. Al-Karaki, and J. Morillo-Pozo, "Wireless Multimedia Sensor Networks: Current Trends and Future Directions," *Sensors Open Access Journal, MDPI International, Basel, Switzerland*, vol. 10, no. 7, pp. 6662–6717, July 2010.

[8] Ambient Systems. [Online]. Available: http://www.ambient-systems.net/

[9] J. Ammer and J. Rabaey, "The Energy-Per-Useful-Bit Metric for Evaluating and Optimizing Sensor Network Physical Layers." International Workshop on Ad Hoc and Sensor Networks (IWWAN), New York, USA, June 2006, pp. 695–700.

[10] G. Anastasi, M. Conti, E. Gregori, and A. Passarella, "802.11 Power-saving Mode for Mobile Computing in Wi-Fi Hotspots: Limitations, Enhancements and Open Issues," *Wireless Networks, Kluwer Academic Publishers, Hingham, MA, USA*, vol. 14, 2008.

[11] T. Andel and A. Yasinsac, "On the Credibility of MANET Simulations," *IEEE Computer Magazine, Los Alamitos, CA, USA*, vol. 39, July 2006.

[12] M. Anwander, G. Wagenknecht, T. Braun, and K. Dolfus, "BEAM: A Burst-Aware Energy-Efficient Adaptive MAC Protocol for Wireless Sensor Networks." International Conference on Networked Sensing Systems (INSS), Kassel, Germany, June 2010, pp. 195–202.

[13] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, "BonnMotion: a Mobility Scenario Generation and Analysis Tool." International Conference on Simulation Tools and Techniques (SimuTools), Malaga, Spain, March 2010, pp. 51:1–51:10.

[14] M. Baar, E. Koeppe, A. Liers, and J. Schiller, "The ScatterWeb MSB-430 Platform for Wireless Sensor Networks." SICS Contiki Workshop, Kista, Sweden, March 2007.

[15] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks." International Conference on Mobile Computing and Networking (MobiCom), Berkeley, USA, November 1995, pp. 2–11.

[16] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, "The Hitchhiker's Guide to successful Wireless Sensor Network Deployments." ACM Conference on Embedded Networked Sensor Systems (SenSys), Raleigh, USA, November 2008, pp. 43–56.

[17] S. Barthlomé, "Investigating Forward Error Correction Strategies on MSB430 Sensor Nodes." Master Thesis, University of Bern, Switzerland, May 2011.

[18] E. R. Berlekamp, *Algebraic Coding Theory, Revised Edition*. McGraw-Hill, New York, 1968.

[19] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Wöhrle, and M. Yücel, "Operating a Sensor Network at 3500 m Above Sea Level." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Demos and Posters, San Francisco, USA, April 2009, pp. 405–406.

[20] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: an Embedded

Multithreaded Operating System for Wireless Micro Sensor Platforms," *ACM/Kluwer Mobile Networks & Applications (MONET), Hingham, USA*, vol. 10, no. 4, pp. 563–579, August 2005.

[21] C. Boano, T. Voigt, N. Tsiftes, L. Mottola, K. Roemer, and M. Zuniga, "Making Sensornet MAC Protocols Robust Against Interference." European Conference on Wireless Sensor Networks (EWSN), Coimbra, Portugal, February 2010, pp. 272–288.

[22] R. Bose and D. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and Control Journal, San Diego, CA, USA*, vol. 3, no. 1, pp. 68 – 79, March 1960.

[23] R. Braden, "Requirements for Internet Hosts - Communication Layers," Request for Comments RFC 1122, Internet Engineering Task Force (IETF), October 1989. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[24] T. Braun, T. Voigt, and A. Dunkels, "TCP Support for Sensor Networks." Wireless On demand Network Systems and Services (WONS), Obergurgl, Austria, January 2007, pp. 162–169.

[25] M. Buettner, V. Gary, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Boulder, USA, November 2006, pp. 307–320.

[26] U. Bürgi, "Performance Optimization for TCP-based Wireless Sensor Networks." Master Thesis, University of Bern, Switzerland, August 2011.

[27] M. Busse, T. Haenselmann, T. King, and W. Effelsberg, "The Impact of Forward Error Correction on Wireless Sensor Network Performance." ACM Workshop on Real-World Wireless Sensor Networks (REALWSN), Uppsala, Sweden, June 2006.

[28] C. Cano, B. Bellalta, A. Sfairopoulou, and J. Barcelo, "A Low Power Listening MAC with Scheduled Wake Up after Transmissions for WSNs," *IEEE Communications Letters*, pp. 221–223, April 2009.

[29] M. Ceriotti, M. Corra, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G. Jesi, R. L. Cigno, L. Mottola, A. Murphy, G. Picco, M. Pescalli, D. Pregnolato, and C. Torghele, "Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Chicago, USA, April 2011, pp. 187–198.

[30] M. Ceriotti, L. Mottola, G. Picco, A. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, "Monitoring Heritage Buildings with Wireless

Sensor Networks: The Torre Aquila Deployment." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), St. Louis, Missouri, USA, April 2008, pp. 277–288.

[31] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks." International Conference on Distributed Computing Systems (ICDCS), Amsterdam, Netherlands, May 1998, pp. 472–479.

[32] M. Chang and P. Bonnet, "Meeting Ecologists' Requirements with Adaptive Data Acquisition." ACM Conference on Embedded Networked Sensor Systems (SenSys), Zurich, Switzerland, November 2010, pp. 141–154.

[33] S. Chatterjea, L. van Hoesel, and P. Havinga, "AI-LMAC: An Adaptive, Information-centric and Lightweight MAC Protocol for Wireless Sensor Networks." International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, December 2004, pp. 381–388.

[34] R. T. Chien, "Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 357–363, October 1964.

[35] O. Chipara, L. Chenyang, C. Thomas, and R. Gruia-Catalin, "Reliable Clinical Monitoring using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit." ACM Conference on Embedded Networked Sensor Systems (SenSys), Zurich, Switzerland, November 2010, pp. 155–168.

[36] Computer Services Department of University of Bern (Informatikdienste), "UBELIX - University of Bern Linux Cluster." [Online]. Available: http://www.id.unibe.ch/content/services/ubelix/

[37] Crossbow Technologies. [Online]. Available: http://www.xbow.com

[38] T. V. Dam and K. Langendoen, "An Adaptive Energy Efficient MAC Protocol for Wireless Sensor Networks (TMAC)." ACM Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, USA, November 2003, pp. 171–180.

[39] Debian Linux - The Universal Linux Operating System. [Online]. Available: http://www.debian.org/releases/lenny/

[40] Deliverable D4.1: First Set of well-designed Simulations, "Experiments and possible Benchmarks. Technical Report," June 2008. [Online]. Available: http://www.wisebed.eu

[41] I. Demirkol, C. Ersoy, and F. Alagoz, "MAC Protocols for Wireless Sensor Networks: A Survey," *IEEE Communications Magazine*, vol. 44, pp. 115–121, April 2006.

[42] DFN: Deutsches Forschungsnetz. [Online]. Available: http://www.dfn.de

[43] N. Draper and H. Smith, "Applied Regression Analysis, Wiley Series in Probability and Statistics," 1998.

[44] W. Drytkiewicz, S. Sroka, V. Handziski, A. Koepke, and H. Karl, "A Mobility Framework for OMNeT++." International OMNeT++ Workshop, co-located with International Conference on Simulation Tools and Techniques (SIMUTools), Budapest, Hungary, January 2003. [Online]. Available: http://mobility-fw.sourceforge.net

[45] A. Dunkels, "Full TCP/IP for 8-Bit Architectures." International Conference on Mobile Systems, Applications, and Services (MobiSys), San Francisco, USA, May 2003, pp. 85–98.

[46] A. Dunkels, J. Alonso, T. Voigt, and H. Ritter, "Distributed TCP Caching for Wireless Sensor Networks." Mediterranean Ad-Hoc Networks Workshop (Med-Hoc-Net), Bodrum, Turkey, June 2004, pp. 13–28.

[47] A. Dunkels, B. Groenvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors." IEEE Workshop on Embedded Networked Sensors (EmNets), Tampa, Florida, November 2004, pp. 455–462. [Online]. Available: http://www.sics.se/contiki/

[48] A. Dunkels, L. Mottola, N. Tsiftes, F. Osterlind, J. Eriksson, and N. Finne, "The Announcement Layer: Beacon Coordination for the Sensornet Stack." European Conference on Wireless Sensor Networks (EWSN), Bonn, Germany, February 2011, pp. 211–226.

[49] A. Dunkels, F. Österlind, and Z. He, "An Adaptive Communication Architecture for Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Sydney, Australia, November 2007, pp. 335–349.

[50] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based On-line Energy Estimation for Sensor Nodes." IEEE Workshop on Embedded Networked Sensors (EmNets), Cork, Ireland, June 2007, pp. 28–32.

[51] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems." ACM Conference on Embedded Networked Sensor Systems (SenSys), Boulder, USA, November 2006, pp. 29–42.

[52] S. Duquennoy, F. Österlind, and A. Dunkels, "Lossy Links, Low Power, High Throughput." ACM Conference on Embedded Networked Sensor Systems (SenSys), Seattle, USA, November 2011.

[53] Dust Networks. [Online]. Available: http://www.dustnetworks.com/

[54] V. Dyo, S. Ellwood, D. MacDonald, A. Markham, C. Mascolo, B. Pasztor, S. Scellato, N. Trigoni, R. Wohlers, and K. Yousef, "Evolution and Sustainability of a Wildlife Monitoring Sensor Network." ACM Conference on Embedded Networked Sensor Systems (SenSys), Zurich, Switzerland, November 2010, pp. 127–140.

[55] EduGAIN - GÉANT Authentication and Authorization Infrastructure (eduGAIN). [Online]. Available: http://www.edugain.org/

[56] A. El-Hoiydi, "Energy Efficient Medium Access Control for Wireless Sensor Networks." PhD Thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2005.

[57] A. El-Hoiydi and J. D. Decotignie, "WiseMAC: An Ultra Low Power MAC Protocol for Multihop Wireless Sensor Networks." International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS), Turku, Finland, July 2004, pp. 18–31.

[58] Elsevier Properties S.A: ScienceDirect Online Digital Library. [Online]. Available: http://www.elsevier.com

[59] J. Eriksson, F. Osterlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt, "COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks." International Conference on Simulation Tools and Techniques (SimuTools), Rome, Italy, March 2009, pp. 27:1–27:7.

[60] E. Ertin, A. Arora, R. Ramnath, and M. Nesterenko, "Kansei: A Testbed For Sensing At Scale." ACM/IEEE International Conference on Information Processing In Sensor Networks (IPSN), Nashville, Tennessee, USA, April 2006, pp. 399–406.

[61] L. Evers, H. J. Kuper, M. E. M. Lijding, and N. Meratnia, "SensorScheme: Supply Chain Management Automation using Wireless Sensor Networks." IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Patras, Greece, September 2007, pp. 448–455.

[62] G. Fairhurst and L. Wood, "Advice to Link Designers on Link Automatic Repeat reQuest (ARQ)," Request for Comments RFC 3366, Internet Engineering Task Force (IETF), August 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3366.txt

[63] L. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment." IEEE International Conference on Computer Communications (INFOCOM), Anchorage, USA, April 2001, pp. 1548–1557.

[64] S. P. Fekete, A. Kroller, S. Fischer, and D. Pfisterer, "Shawn: The fast, highly customizable Sensor Network Simulator." International Conference on Networked Sensing Systems (INSS), Braunschweig, Germany, June 2007, pp. 299–308.

[65] N. Finne, J. Eriksson, N. Tsiftes, and A. D. T. Voigt, "Improving Sensornet Performance by Separating System Configuration from System Logic." European Conference on Wireless Sensor Networks (EWSN), Coimbra, Portugal, February 2010, pp. 195–209.

[66] Freescale Corporation: MMA7260Q XYZ Three-Axis Low g Acceleration Sensor. [Online]. Available: www.freescale.com

[67] T. Gao, T. Massey, L. Selavo, D. Crawford, C. Borrong, K. Lorincz, V. Shnayder, L. Hauenstein, F. Dabiri, J. Jeng, A. Chanmugam, D. White, M. Sarrafzadeh, and M. Welsh, "The Advanced Health and Disaster Aid Network: A Light-weight Wireless Medical System for Triage," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 3, pp. 203–216, January 2007.

[68] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multihop Networks." IEEE Workshop on Mobile Computer Systems and Applications (HotMobile), New Orleans, USA, February 1999, pp. 41–50.

[69] T. A. Gulliver and V. K. Bhargava, "A Systematic (16,8) Code for Correcting Double Errors and Detecting Triple-Adjacent Errors," *IEEE Transactions on Computers*, vol. 42, January 1993.

[70] C. Guo, L. Zhong, and J. Rabaey, "Low Power Distributed MAC for Ad Hoc Sensor Radio Networks." IEEE Global Telecommunications Conference (GLOBECOM), San Antonio, USA, November 2001, pp. 2944–2948.

[71] G. Halkes and K. Langendoen, "Crankshaft: An Energy-Efficient MAC-Protocol For Dense Wireless Sensor Networks." European Conference on Wireless Sensor Networks (EWSN), Delft, Netherlands, January 2007, pp. 228–244.

[72] G. P. Halkes, T. van Dam, and K. G. Langendoen, "Comparing Energy-Saving MAC Protocols for Wireless Sensor Networks," *ACM/Kluwer Mobile Networks & Applications (MONET), Hingham, USA*, vol. 10, pp. 783–791, October 2005.

[73] Hameg Instruments - HM1508-2 150 MHz Mixed Signal CombiScope with FFT. [Online]. Available: http://www.hameg.com/322.0.html

[74] R. Hamming, "Error Detecting and Error Correcting Codes," vol. 26, no. 2, pp. 147–160, April 1950.

[75] V. Handziski, A. Koepke, A. Willig, and A. Wolisz, "TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Network." ACM/SIGMOBILE International Workshop on Multi-hop Ad Hoc Networks (REALMAN), Florence, Italy, May 2006.

[76] I. Haratcherev, G.Halkes, T.Parker, O.Visser, and K.Langendoen, "PowerBench: a Scalable Testbed Infrastructure for Benchmarking Power Consumption." International Workshop on Sensor Network Engineering (IWSNE), Santorini, Greece, June 2008, pp. 37–44.

[77] A. Hergenroeder, J. Horneber, D. Meier, P. Armbruster, and M. Zitterbart, "Distributed Energy Measurements in Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Demo Session, Berkeley, USA, November 2009.

[78] A. Hergenroeder, J. Wilke, and D. Meier, "Distributed Energy Measurements in WSN Testbeds with a Sensor Node Management Device (SNMD)." International Conference on Architecture of Computing Systems (ARCS), Hannover, Germany, February 2010, pp. 341–348.

[79] A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres (Paris) 2*, pp. 147–156, September 1959.

[80] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks." ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), Seattle, Washington, USA, August 1999, pp. 219–230.

[81] J. Hong and M. Vetterli, "Simple Algorithms for BCH Decoding," *IEEE Transactions on Communications*, vol. 43, no. 8, pp. 2324–2333, August 1995.

[82] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating Congestion in Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, USA, November 2004, pp. 134–147.

[83] P. Hurni, M. Anwander, and G. Wagenknecht, "TARWIS - Testbed Management Architecture for Wireless Sensor Network Testbeds." [Online]. Available: http://rvs.unibe.ch/research/software.html

[84] P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, and T. Braun, "TARWIS - A Testbed Management Architecture for Wireless Sensor Network Testbeds." International Conference on Network and Service Management (CNSM), Short Paper Session, Paris, France, October 2011, pp. 1–5.

[85] P. Hurni and T. Braun, "MaxMAC: a Maximally Traffic-Adaptive MAC Protocol for Wireless Sensor Networks." European Conference on Wireless Sensor Networks (EWSN), Coimbra, Portugal, February 2010, pp. 289–305.

[86] ——, "On the Accuracy of Software-based Energy Estimation Techniques." European Conference on Wireless Sensor Networks (EWSN), Bonn, Germany, February 2011, pp. 49–64.

[87] ——, "Increasing Throughput for WiseMAC." IEEE/IFIP Wireless On demand Network Systems and Services (WONS), Garmisch-Partenkirchen, Germany, January 2008, pp. 105–108.

[88] ——, "Real-World Experiences with the Maximally Traffic Adaptive Medium Access Control Protocol." Technical Report IAM-11-001, Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, June 2011.

[89] ——, "Evaluation of WiseMAC and Extensions on Wireless Sensor Nodes," *Springer Telecommunication Systems Journal*, vol. 43, no. 1-2, pp. 49–58, March 2010.

[90] ——, "On the Adaptivity of Today's Energy-Efficient MAC Protocols under varying Traffic Conditions." IEEE Conference on Ultra-Modern Technologies (ICUMT), St. Petersburg, Russia, October 2009, pp. 43–51.

[91] ——, "Evaluation of WiseMAC on Sensor Nodes." IFIP International Conference on Mobile and Wireless Communications Networks (MWCN), Toulouse, France, September 2008, pp. 187–198.

[92] P. Hurni, G.Wagenknecht, M. Anwander, and T. Braun, "A Testbed Management System for Wireless Sensor Network Testbeds (TARWIS)." European Conference on Wireless Sensor Networks (EWSN) Demo Session, Coimbra, Portugal, February 2010, pp. 33–35.

[93] P. Hurni and S. Barthlomé, "libECC: An Open-Source Library of Error Correcting Codes (ECCs) for the MSP430 Microcontroller." [Online]. Available: http://rvs.unibe.ch/research/software.html

[94] P. Hurni, S. Barthlomé, and T. Braun, "Link-Quality Aware Run-Time Adaptive Forward Error Correction Strategies in Wireless Sensor Networks." Technical Report IAM-11-003, Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, February 2012.

[95] P. Hurni and T. Braun, "Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments." International IFIP Networking Conference (IFIP NETWORKING), Aachen, Germany, May 2009, pp. 1–13.

[96] P. Hurni, U. Bürgi, T. Braun, and M. Anwander, "Performance Optimizations for TCP in Wireless Sensor Networks." European Conference on Wireless Sensor Networks (EWSN), Trento, Italy, February 2012.

[97] IEEE Xplore: The IEEE Xplore Digital Library. [Online]. Available: http://ieeexplore.ieee.org

[98] Javier Solobera, "Libelium Technologies Incorporated - Detecting Forest Fires using Wireless Sensor Networks with Waspmotes," 2010. [Online]. Available: http://www.libelium.com/libeliumworld/articles/101031032811

[99] J. Jeong and C. T. Ee, "Forward Error Correction in Sensor Networks." International Workshop on Wireless Sensor Networks (WWSN), Marrakesh, Morocco, June 2007.

[100] J. Kahn, R. Katz, and K. Pister, "Next Century Challenges: Mobile Networking for 'Smart Dust'." ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom), Seattle, USA, August 1999, pp. 271–278.

[101] M. Karagiannis, I. Chatzigiannakis, and J. Rolim, "WSNGE: A Platform for simulating complex Wireless Sensor Networks supporting rich Network Visualization and Online Interactivity." Spring Simulation Multiconference (SpringSim), San Diego, USA, March 2009, pp. 35:1–35:8.

[102] C. Kompis and S. Aliwell, "Energy Harvesting Technologies to Enable Wireless and Remote Sensing." Sensors & Instrumentation KTN Action Group Report, Vodera Ltd. and Zartech Ltd., June 2008.

[103] D. Kotz, C. Newport, R. Gray, J. Liu, Y. Yuan, and C. Elliott, "Experimental Evaluation of Wireless Simulation Assumptions." International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Venice, Italy, October 2004, pp. 78–82.

[104] A. Kuntz, F. Schmidt-Eisenlohr, O. Graute, H. Hartenstein, and M. Zitterbart, "Introducing Probabilistic Radio Propagation Models in OMNeT++ MF and Cross Validation Check with NS-2." International Workshop on OMNeT++, co-located with International Conference on Simulation Tools and Techniques (SIMUTools), Marseille, France, March 2008.

[105] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET Simulation Studies: the Incredibles," *ACM/SIGMOBILE Mobile Computer Communications Revue, New York, USA*, vol. 9, no. 4, pp. 50–61, October 2005.

[106] O. Landsiedel, K. Wehrle, and S. Goetz, "Accurate Prediction of Power Consumption in Sensor Networks." IEEE Workshop on Embedded Networked Sensors (EmNets), Sydney, Australia, May 2005, pp. 37–44.

[107] K. Langendoen and A. Meier, "Analyzing MAC Protocols for Low Data-Rate Applications," *ACM Transactions on Sensor Networks (TOSN), New York, USA*, vol. 7, no. 2, pp. 1–40, August 2010.

[108] K. Langendoen, "The MAC Alphabet Soup, an Index into various Medium Access Control (MAC) Protocols." University of Delft, Netherlands. [Online]. Available: http://www.st.ewi.tudelft.nl/koen/MACsoup/

[109] ——, "Medium Access Control in Wireless Sensor Networks," *Nova Science Publishers, New York, USA*, pp. 535–560, May 2008, Bookchapter.

[110] D.-U. Lee, H. Kim, S. Tu, M. H. Rahimi, D. Estrin, and J. D. Villasenor, "Energy-Optimized Image Communication on Resource-constrained Sensor Platforms." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Sydney, Australia, November 2007, pp. 216–225.

[111] S. Lee, J. Park, and L. Choi, "AMAC: Traffic-Adaptive Sensor Network MAC Protocol through Variable Duty-Cycle Operations." IEEE International Conference on Communications (ICC), Glasgow, Scotland, June 2007, pp. 3259–3264.

[112] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks." Ambient Intelligence Part II, Springer Verlag, USA, 2004, pp. 115–148.

[113] Q. Li, F. Osterlind, T. Voigt, S. Fischer, and D. Pfisterer, "Making Wireless Sensor Network Simulators Cooperate." ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN), Bodrum, Turkey, October 2010, pp. 95–98.

[114] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis, "Surviving Wi-fi Interference in Low Power ZigBee Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Zurich, Switzerland, November 2010, pp. 309–322.

[115] S. Lin and J. D. J. Costello, *Error Control Coding: Fundamentals and Applications Second Edition*. Prentice Hall: Englewood Cliffs, NJ, 2004.

[116] S. Ling and C. Xing, *Coding Theory: A First Course*. Cambridge University Press, 2004.

[117] Linksys Network Storage Link for USB 2.0 Disk Drives. [Online]. Available: http://www.linksysbycisco.com/UK/en/products/NSLU2

[118] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," September 2002, pp. 88–97.

[119] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care." MobiSys Workshop on Applications of Mobile Embedded Systems (WAMES), Boston, USA, June 2004.

[120] J. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, January 2003.

[121] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," Request for Comments RFC 2018, Internet Engineering Task Force (IETF), October 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[122] Mathpages.com, "A Collection of Lectures on various Subjects in Mathematics and Physics: Cyclic Redundancy Checksums (CRCs)." [Online]. Available: http://www.mathpages.com/home/kmath458.htm

[123] The Swiss Education and Research Network (SWITCH), "Authentication and Authorization Infrastructure: System and Interface Specification." [Online]. Available: http://www.switch.ch/aai/demo/

[124] A. Meier, M. Wöhrle, M. Zimmerling, and L. Thiele, "Zerocal: Automatic MAC Protocol Calibration." Distributed Computing in Sensor Systems (DCOSS), Santa Barbara, USA, June 2010, pp. 31–44.

[125] Micro Strain. [Online]. Available: http://www.microstrain.com/

[126] T. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[127] R. Morelos-Zaragoza, *The Art of Error Correcting Coding*. Second Edition, John Wiley and Sons, 2006.

[128] MoteWeb: Harvard Sensor Network Testbed Software. [Online]. Available: http://motelab.eecs.harvard.edu/

[129] mspgcc - A port of the GNU tools to the Texas Instruments MSP430 microcontrollers. [Online]. Available: http://mspgcc.sourceforge.net

[130] R. Musaloiu, C. Liang, and A. Terzis, "Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), St. Louis, USA, April 2008, pp. 421–432.

[131] MySQL Database: Open Source Database Management System. [Online]. Available: http://www.mysql.com/

[132] A. Nguyen, A. Foerster, D. Puccinelli, and S. Giordano, "Sensor Node Lifetime: An Experimental Study." IEEE International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSens), Lugano, Switzerland, March 2011, pp. 202–207.

[133] "The Network Simulator NS-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[134] NSLU2-Linux: Linux Distributions for the Linksys NS2LU. [Online]. Available: http://www.nslu2-linux.org/

[135] Numonyx Forte Serial Flash Memory M25P80: 8 Mbit, low voltage, serial Flash memory with 75 MHz SPI bus interface. [Online]. Available: http://numonyx.com/Documents/Datasheets/M25P80.pdf

[136] B. Nyffenegger, "Evaluation and Calibration of Software-based Energy Estimation Methodologies on MSB430 Sensor Nodes." Master Thesis, University of Bern, Switzerland, December 2010.

[137] F. Österlind and A. Dunkels, "Approaching the Maximum 802.15.4 Multihop Throughput." ACM Workshop on Embedded Networked Sensors (HotEmNets), Charlottesville, USA, June 2008.

[138] B. Pasztor, L. Mottola, C. Mascolo, G. Picco, S. Ellwood, and D. MacDonald, "Selective Reprogramming of Mobile Sensor Networks through Social Community Detection." European Conference on Wireless Sensor Networks (EWSN), Coimbra, Portugal, February 2010, pp. 178–193.

[139] PC Engines GmbH, "ALIX System Boards." [Online]. Available: www.pcengines.ch

[140] PC Engines GmbH: Wireless Router Application Platform (WRAP). [Online]. Available: www.pcengines.ch

[141] W. Peterson and D. Brown, "Cyclic Codes for Error Detection," *Proceedings of the Institute of Electrical and Electronic Engineers (IRE)*, vol. 49, no. 1, January 1961.

[142] PHP: Hypertext Preprocessor. [Online]. Available: http://www.php.net/

[143] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, USA, November 2004, pp. 95–107.

[144] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research." International Conference on Information Processing in Sensor Networks (IPSN), Los Angeles, USA, April 2005, pp. 364–369.

[145] J. Postel, "Simple Mail Transfer Protocol," Request for Comments RFC 851, Internet Engineering Task Force (IETF), August 1982. [Online]. Available: http://www.ietf.org/rfc/rfc851.txt

[146] ——, "File Transfer Protocol," Request for Comments RFC 765, Internet Engineering Task Force (IETF), October 1985. [Online]. Available: http://www.ietf.org/rfc/rfc765.txt

[147] ——, "Transmission Control Protocol (TCP)," Request for Comments RFC 793, Internet Engineering Task Force (IETF), September 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[148] J. Postel and J. Reynolds, "Telnet Protocol Specification," Request for Comments RFC 854, Internet Engineering Task Force (IETF), May 1983. [Online]. Available: http://www.ietf.org/rfc/rfc854.txt

[149] G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors," *Communications of the ACM, New York, USA*, vol. 43, pp. 51–58, May 2000.

[150] M. H. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), San Diego, USA, November 2005, pp. 192–204.

[151] V. Rajendran, K. Obraczka, and J. Garcia-Luna-Aceves, "Energy-Efficient and Collision-Free Medium Access Control for Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, USA, November 2003, pp. 181–192.

[152] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale, "PIP: a Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer." ACM Conference on Embedded Networked Sensor Systems (SenSys), Zurich, Switzerland, November 2010, pp. 15–28.

[153] T. S. Rappaport, "Wireless Communications: Principles & Practise." Prentice Hall, Upper Saddle River, NJ, 2nd Edition, 2002.

[154] I. Reed and G. Solomon, "Polynomial Codes over certain finite Fields," *Society for Industrial and Applied Mathematics (SIAM) Journal*, vol. 8, no. 2, pp. 300–304, June 1960.

[155] O. Rensfelt, F. Hermans, L. Larzon, and P. Gunningberg, "Sensei-UU: a relocatable Sensor Network Testbed." ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH), Chicago, USA, September 2010, pp. 63–70.

[156] RF Monolithics Incorporated: TR1001 868.35 MHz Hybrid Transceiver. [Online]. Available: www.rfm.com/products/data/tr1001.pdf

[157] ScatterWeb^2 Operating System - Freie Universität Berlin & ScatterWeb GmbH. [Online]. Available: http://scatterweb.mi.fu-berlin.de/

[158] J. Schiller, A. Liers, H. Ritter, R. Winter, and T. Voigt, "ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing." Hawaii International Conference on System Sciences (HICSS), Hawaii, USA, January 2005, pp. 1–9.

[159] J. H. Schiller, A. Liers, and H. Ritter, "ScatterWeb: A Wireless Sensornet Platform for Research and Teaching," *Elsevier Computer Communications*, vol. 28, pp. 1545–1551, August 2005.

[160] C. Schurgers, V. Tsiatsis, and M. B. Srivastava, "STEM: Topology Management for Energy Efficient Sensor Networks." IEEE Aerospace Conference, Big Sky, USA, November 2002, pp. 78–89.

[161] Sensirion Sensor Company: SHT11 Digital Humidity and Temperature Sensor. [Online]. Available: www.sensirion.com/sht11

[162] Seventh Framework Programme FP7 - Information and Communication Technologies, "Wireless Sensor Networks Testbed Project (WISEBED)," FP7 Project 2008-2011. [Online]. Available: http://www.wisebed.eu

[163] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, October 1948.

[164] Shibboleth: A Standards-based Open-Source Internet2 Middleware Architecture Project. [Online]. Available: http://shibboleth.internet2.edu

[165] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton, "Sensor Network-Based Countersniper System." ACM Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, USA, November 2004, pp. 1–12.

[166] S. Singh and C. S. Raghavendra, "PAMAS: Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks," *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 5–26, July 1998.

[167] SOWNet Technologies B.V. [Online]. Available: http://www.sownet.nl

[168] T. Starr, J. M. Cioffi, and P. J. Silverman, *Understanding Digital Subscriber Line Technology*. Prentice Hall, Upper Saddle River, USA, 1999.

[169] T. Staub, D. Balsiger, M. Lustenberger, and T. Braun, "Secure Remote Management and Software Distribution for Wireless Mesh Networks." International Workshop on Applications and Services in Wireless Networks (ASWN), Santander, Spain, May 2007, pp. 47–54.

[170] P. Suarez, C. Renmarker, T. Voigt, and A. Dunkels, "Increasing ZigBee network lifetime with X-MAC." ACM Workshop on Real-World Wireless Sensor Network (REALWSN), Glasgow, Scotland, November 2008, pp. 13–18.

[171] Texas Instruments, "16-Bit Ultra-Low Power MSP430 Microcontrollers: Datasheets, Specifications and Reference Manuals." [Online]. Available: http://focus.ti.com/

[172] Texas Instruments CC1000: Single Chip Very Low Power RF Transceiver. [Online]. Available: http://www.ti.com/lit/gpn/cc1000

[173] Texas Instruments CC1020: Single-Chip FSK/OOK CMOS RF Transceiver. [Online]. Available: http://www.ti.com/lit/gpn/cc1020

[174] Texas Instruments CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. [Online]. Available: http://focus.ti.com/lit/ds/symlink/cc2420

[175] The World Wide Web Consortium (W3C): Web Services Activity Group. [Online]. Available: http://www.w3.org/2002/ws/

[176] Tutornet University of Southern California, "A Tiered Wireless Sensor Network Testbed." [Online]. Available: http://enl.usc.edu/projects/tutornet

[177] TWISTv1: The TWIST testbed software suite, 2011. [Online]. Available: http://www.twist.tu-berlin.de/wiki/TWIST/Software/TWISTv1

[178] A. Varga, "The OMNeT++ Discrete Event Simulation System." European Simulation Multiconference (ESM), Prague, Czech Republic, June 2001, pp. 319–324. [Online]. Available: http://www.omnetpp.org

[179] M. Wälchli, P. Skoczylas, M. Meer, and T. Braun, "Building Intrusion Detection with a Wireless Sensor Network." ICST International Conference on Ad Hoc Networks (AdHocNets), Niagara Falls, Canada, September 2009, pp. 607–622.

[180] C. Wan and S. Eisenman, "CODA: Congestion Detection and Avoidance in Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, USA, November 2003, pp. 266–279.

[181] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American (SciAm)*, vol. 265, no. 3, pp. 94–104, September 1991.

[182] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a Wireless Sensor Network Testbed." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Los Angeles, USA, April 2005, pp. 483–488.

[183] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and Yield in a Volcano Monitoring Sensor Network." USENIX Symposium on Operating Systems Design and Implementation (OSDI), Seattle, USA, November 2006, pp. 27–39.

[184] Wikipedia - The Free Encyclopedia, "Majority Logic Decoding for Repetition Coding," 2010. [Online]. Available: http://en.wikipedia.org/wiki/Majority_logic_decoding

[185] A. Willig and R. Mitschke, "Results of Bit Error Measurements with Sensor Nodes and casuistic Consequences for Design of Energy-Efficient Error Control Schemes." European Workshop on Sensor Networks (EWSN), Zurich, Switzerland, February 2006, pp. 310–325.

[186] G. Wittenburg, N. Dziengel, C. Wartenburger, and J. Schiller, "A System for Distributed Event Detection in Wireless Sensor Networks." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Stockholm, Sweden, April 2010, pp. 94–104.

[187] S. Yanjun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A Receiver Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Raleigh, USA, November 2008, pp. 1–14.

[188] S. Yanjun, D. Shu, O. Gurewitz, and D. Johnson, "DW-MAC: a Low Latency, Energy Efficient Demand-Wakeup MAC protocol for Wireless Sensor Networks." ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Hong Kong, China, May 2008, pp. 53–62.

[189] W. Ye and J. Heidemann, "Medium Access Control in Wireless Sensor Networks." Kluwer Academic Publishers, Norwell, Massachusetts, USA, 2004, pp. 73–91, Bookchapter.

[190] W. Ye, J. Heidemann, and D. Estrin, "An Energy Efficient MAC Protocol for Wireless Sensor Networks." IEEE International Conference on Computer Communications (INFOCOM), New York, USA, June 2002, pp. 1567–1576.

[191] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, USA, November 2003, pp. 192–204.

[192] R. Zheng and R. Kravets, "On-Demand Power Management for Ad Hoc Networks." IEEE International Conference on Computer Communications (INFOCOM), San Francisco, USA, April 2003, pp. 481–491.

List of Publications

Refereed Papers (Journals, Conferences, Workshops)

- Philipp Hurni, Ulrich Bürgi, Torsten Braun, Markus Anwander: Performance Optimizations for TCP in Wireless Sensor Networks, accepted for publication at *European Conference on Wireless Sensor Networks (EWSN)*, Trento, Italy, February 2012
- P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, T. Braun: TARWIS - A Testbed Management Architecture for Wireless Sensor Network Testbeds, *International Conference on Network and Service Management (CNSM)*, Paris, France, October 2011
- P. Hurni, B. Nyffenegger, T. Braun, A. Hergenroeder: On the Accuracy of Software-based Energy Estimation Techniques, *European Conference on Wireless Sensor Networks (EWSN)*, Bonn, Germany, February 2011
- P. Hurni, T. Braun: MaxMAC: A Maximally Traffic-Adaptive MAC Protocol for Wireless Sensor Networks, *European Conference on Wireless Sensor Networks (EWSN)*, Coimbra, Portugal, February 2010
- P. Hurni, T. Braun: An Energy-Efficient Broadcasting Scheme for Unsynchronized Wireless Sensor MAC Protocols, *IEEE Conference on Wireless On-demand Network Systems and Services (WONS)*, Kranjska Gora, Slovenia, February 2010
- Z. Yu, B. Bhargava, P. Hurni: The Effects of Threading, Infection Time, and Multiple-Attacker Collaboration on Malware Propagation, *IEEE International Symposium on Reliable Distributed Systems (SRDS)*, New York, USA, September 2009
- P. Hurni, T. Braun: Evaluation of WiseMAC and Extensions on Sensor Nodes, *Springer Telecommunication Systems Journal, Vol. 43, Nr. 1-2*, September 2009, Springer US, ISSN 1018-4864
- P. Hurni, T. Braun: On the Adaptivity of Today's Energy-Efficient MAC Protocols under varying Traffic Conditions, *IEEE Conference on Ultra-Modern Technologies (ICUMT)*, St. Petersburg, Russia, October 2009
- P. Hurni, T. Staub, G. Wagenknecht, M. Anwander, T. Braun: A Secure Remote Authentication, Operation and Management Infrastructure for Distributed Wireless Sensor Network Testbeds, *First Workshop on Global Sen-*

sor Networks (GSN), co-located with KiVS, Kassel, Germany, February 2009

- P. Hurni: Unsynchronized Energy-Efficient MAC and Routing in Wireless Sensor Networks *Praxis der Informationsverarbeitung und Kommunikation, Vol. 09, Nr. 1*, January, 2009, pp. 38-44, K. G. Saur, ISSN 0930-5157
- P. Hurni, T. Braun: Evaluation of WiseMAC on Sensor Nodes, *IFIP International Conference on Mobile and Wireless Communications Networks (MWCN)*, Toulouse, France, September 2008
- P. Hurni, T. Braun: Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments, *IFIP Networking 2009*, Aachen, Germany, May 2009
- P. Hurni, T. Braun, B. Bhargava, Y. Zhang: Multi-Hop Cross-Layer Design in Wireless Sensor Networks: A Case Study, *IEEE Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Avignon, France, October 2008
- P. Hurni, T. Braun: Evaluation of WiseMAC on Sensor Nodes, *IFIP International Conference on Mobile and Wireless Communications Networks (MWCN)*, Toulouse, France, September 2008
- P. Hurni, T. Braun: Energy-Efficient Multi-Path Routing in Wireless Sensor Networks, *International Conference on Ad Hoc Networks & Wireless (AdHoc-NOW)*, Sophia Antipolis, France, September 2008
- P. Hurni, T. Braun: Increasing Throughput for WiseMAC, *IEEE International Conference on Wireless On-Demand Network Systems and Services (WONS)*, Garmisch, Germany, January 2008
- Philipp Hurni, Torsten Braun: Improving Unsynchronized MAC Mechanisms in Wireless Sensor Networks, *ERCIM Workshop on eMobility*, Coimbra, Portugal, May 2007
- P. Hurni, T. Braun L. M. Feeney: Simulation and Evaluation of Unsynchronized Power Saving Mechanisms in Wireless Ad Hoc Networks *International Conference on Wired/Wireless Internet Communications (WWIC)*, Bern, Switzerland, May 2006

Unrefereed Papers (Technical Reports, Project Deliverables)

- P. Hurni, S. Barthlome, T. Braun: Link-Quality Aware Run-Time Adaptive Forward Error Correction Strategies in Wireless Sensor Networks, *Technical Report IAM-11-002*, Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, November 2011
- P. Hurni, T. Braun: Real-World Experiences with the Maximally Traffic Adaptive Medium Access Control Protocol. *Technical Report IAM-11-001*, Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland, June 2011

- M. Gunes, P. Hurni, et al.: Hardware Capabilities, Maintenance and Future Dimensions, *WISEBED Deliverable D1.5*, May 2011
- K. Langendoen, P. Hurni, et al.: Compendium of Experimental Scenarios, Traces and Benchmarks *WISEBED Deliverable D4.3*, May 2011
- S. Fischer, D. Pfisterer, P. Hurni et al: Final Report on Dissemination and Joint Research Activities, *WISEBED Deliverable D5.5*, May 2011
- K. Langendoen, P. Hurni, et al.: Second Set of Real Test-Bed Experiments and Simulation Scenarios, *WISEBED Deliverable D4.2*, June 2010
- S. Fischer, P. Hurni, et al.: Second Report on Dissemination and Joint Research Activities *WISEBED Deliverable D5.2*, June 2010
- S. Fischer, M. Gunes, G. Coulson, S. Fekete, K. Langendoen, P. Hurni, et al.: Design of the Hardware Infrastructure, Architecture of the Software Infrastructure, and Design of Library of Algorithms *WISEBED Deliverable D1.1, D2.1, D3.1*, June 2009
- M. Brogle, S. Serbu, D. Milic, M. Anwander, P. Hurni, C. Spielvogel, C. Fautsch, D. Harmanci, L. Charles, H. Sturzrehm, G.Wagenknecht, T. Braun, T. Staub, C. Latze, R. Standtke: *BeNeFri Universities Summer School on Dependable Systems*, Schloss Münchenwiler, Switzerland, September 2009, IAM-09-006
- M. Gunes, P. Hurni, et al.: Initial Hardware Installation *WISEBED Deliverable D1.2*, June 2009
- G. Coulson, B. Porter, P. Hurni, et al: Report on the Implementation of the Software Infrastructure *WISEBED Deliverable D2.2*, June 2009
- S. Fischer, D. Pfisterer, P. Hurni et al: Design of the Hardware Infrastructure, Architecture of the Software Infrastructure, and Design of Library of Algorithms *WISEBED Deliverable D1.1, D2.1, and D3.1*, December 2008
- M. Brogle, D. Milic, M. Anwander, G. Wagenknecht, M. Waelchli, T. Braun, R. Kummer, M. Wulff, R. Standtke, H. Sturzrehm, E. Riviere, P. Felber, S. Krenn, C. Ehret, C. Latze, P. Hurni, and T. Staub: *BeNeFri Universities Summer School on Dependable Systems*, Quarten, Switzerland, November 2008, IAM-08-003

Curriculum Vitae

Personal Details

Name	Philipp Hurni
Date of Birth	July 29, 1982
Address	Könizstrasse 28
	CH-3008 Bern, Switzerland
Hometown	Fräschels
Nationality	Swiss

Education

2008-2011	PhD Student & Researcher in Computer Science Computer Networks and Distributed Systems Group University of Bern, Switzerland
January-March 2008	Visiting Researcher, Purdue University, Indiana, USA
2007	Master of Science in Computer Science (summa cum laude) University of Bern, Switzerland
2003-2006	Bachelor of Science in Computer Science with Minors in Economics and Mathematics (in signis cum laude) University of Bern, Switzerland
1997-2001	Mathematisch-Naturwissenschaftliches Gymnasium Bern-Neufeld, Emphasis on Physics and Applied Mathematics

Awards

2009	Kommunikation und Verteilte Systeme (KuVS) Award for an outstanding Master Thesis The German Computer Science Society (GI)
2008	Best Master Thesis in Computer Science / Mathematics Faculty of Natural Sciences and Exact Sciences, University of Bern, Switzerland

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: Hurni Philipp

Matrikelnummer: 02-116-382

Studiengang: Informatik

Bachelor ☐ Master ☐ Dissertation ☒

Titel der Arbeit: Traffic-Adaptive and Link-Quality-Aware Communication in Wireless Sensor Networks

LeiterIn der Arbeit: Prof. Dr. Torsten Braun

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 21. November 2011

Ort/Datum

.....

Unterschrift