Algorithms for Energy Efficiency in Wireless Sensor Networks

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Dipl.-Wirtsch.-Inf. Marcel Busse
aus Gehrden

Mannheim, 2007

Dekan:	Professor Dr. Matthias Krause, Universität Mannheim
Referent:	Professor Dr. Wolfgang Effelsberg, Universität Mannheim
Korreferent:	Professor Dr. Colin Atkinson, Universität Mannheim

Tag der mündlichen Prüfung: 21.01.2008

To Katharina.

Abstract

The recent advances in microsensor and semiconductor technology have opened a new field within computer science: the networking of small-sized sensors which are capable of sensing, processing, and communicating. Such *wireless sensor networks* offer new applications in the areas of habitat and environment monitoring, disaster control and operation, military and intelligence control, object tracking, video surveillance, traffic control, as well as in health care and home automation. It is likely that the deployed sensors will be battery-powered, which will limit the energy capacity significantly. Thus, energy efficiency becomes one of the main challenges that need to be taken into account, and the design of energy-efficient algorithms is a major contribution of this thesis.

As the wireless communication in the network is one of the main energy consumers, we first consider in detail the characteristics of wireless communication. By using the *embedded sensor board* (ESB) platform recently developed by the Free University of Berlin, we analyze the means of forward error correction and propose an appropriate resync mechanism, which improves the communication between two ESB nodes substantially. Afterwards, we focus on the forwarding of data packets through the network. We present the algorithms *energy-efficient forwarding* (EEF), *lifetime-efficient forwarding* (LEF), and *energy-efficient aggregation forwarding* (EEAF). While EEF is designed to maximize the number of data bytes delivered per energy unit, LEF additionally takes into account the residual energy of forwarding nodes. In so doing, LEF further prolongs the lifetime of the network. Energy savings due to data aggregation and in-network processing are exploited by EEAF.

Besides *single-link forwarding*, in which data packets are sent to only one forwarding node, we also study the impact of *multi-link forwarding*, which exploits the broadcast characteristics of the wireless medium by sending packets to several (potential) forwarding nodes. By actively selecting a forwarder among all nodes that received a packet successfully, retransmissions can often be avoided. In the majority of cases, multi-link forwarding is thus more efficient and able to save energy.

In the last part of this thesis, we present a *topology and energy control algorithm* (TECA) to turn off the nodes' radio transceivers completely in order to avoid idle listening. By means of TECA, a connected backbone of active nodes is established, while all other nodes may sleep and save energy by turning off their radios. All algorithms presented in this thesis have been fully analyzed, simulated, and implemented on the ESB platform. They are suitable for several applications scenarios and can easily be adapted even to other wireless sensor platforms.

Zusammenfassung

Die jüngsten Fortschritte in der Mikrosensorik und Halbleitertechnik haben einen neuen Forschungsbereich in der Informatik hervorgebracht: die Vernetzung kleinster Sensoren, die in der Lage sind, Daten zu detektieren, zu verarbeiten und zu übertragen. Solche *drahtlosen Sensornetze* ermöglichen neue Anwendungen sowohl für die Beobachtung von biologischen Lebensräumen, den Katastrophenschutz, für militärische Einsätze, der Objektverfolgung und Videoüberwachung, der Verkehrssteuerung, als auch der medizinischen Versorgung und Hausautomation. Dabei ist anzunehmen, dass die eingesetzten Sensoren batteriebetrieben sein werden, wodurch die zur Verfügung stehende Energie stark eingeschränkt sein wird. Energieeffizienz wird somit zu einer der wichtigsten Herausforderungen, und ein wesentlicher Beitrag dieser Dissertation ist der Entwurf von energieeffizienten Algorithmen.

Da die drahtlose Kommunikation in einem Sensornetz mit am meisten Energie verbraucht, betrachten wir zunächst deren Eigenschaften. Unter Verwendung der *Embedded Sensor Board* (ESB) Plattform, die vor kurzem an der Freien Universität Berlin entwickelt wurde, untersuchen wir die Verwendung von Vorwärtsfehlerkorrektur und entwickeln einen geeigneten Resynchronisations-Mechanismus, der die Kommunikation zwischen zwei ESB-Knoten wesentlich verbessert. Anschließend betrachten wir die Weiterleitung von Datenpaketen in einem Sensornetz. Hierfür entwerfen wir die drei Algorithmen *Energy-Efficient Forwarding* (EEF), *Lifetime-Efficient Forwarding* (LEF) und *Energy-Efficient Aggregation Forwarding* (EEAF). Während der EEF-Algorithmus ausschließlich die Anzahl an zugestellten Datenbytes pro verbrauchter Energieeinheit maximiert, berücksichtigt der LEF-Algorithmus zusätzlich die verbleibende Restenergie weiterleitender Sensorknoten. Auf diese Weise kann die Lebenszeit des Netzes weiter verlängert werden. Energieeinsparungen aufgrund von Datenaggregation und der Vorverarbeitung im Inneren des Netzes berücksichtigt das EEAF-Verfahren.

Neben dem so genannten *Single-Link Forwarding*, bei dem Datenpakete ausschließlich zu einen einzigen weiterleitenden Knoten geschickt werden, untersuchen wir auch das so genannte *Multi-Link Forwarding*, das die Eigenschaften des drahtlosen und gemeinsam genutzten Funkmediums ausnutzt, indem Pakete zu mehreren (potenziell) weiterleitenden Knoten geschickt werden. Durch aktives Auswählen eines Knotens, der ein Paket erfolgreich empfangen konnte, können erneute Übertragungen von Datenpaketen häufig vermieden werden. Multi-Link Forwarding erzielt somit meistens eine verbesserte Energieeffizienz und verringert den Energieverbrauch.

Der letzte Teil der Dissertation beschreibt einen *Topology and Energy Control Algorithm* (TECA), der es ermöglicht, die Funkeinheit eines Sensorknotens vollständig auszuschalten, wodurch das so

genannte *Idle Listening* vermieden werden kann. TECA baut ein verbundenes Basisnetz bestehend aus aktiven Knoten auf, während alle anderen Knoten in einen energiesparenden Schlafmodus wechseln können, indem die Funkeinheit ausgeschaltet wird. Alle hier vorgestellten Algorithmen wurden vollständig untersucht, simuliert und auf der ESB-Plattform implementiert. Sie eignen sich für eine Vielzahl von Anwendungszenarien und können leicht auch auf andere Sensor-Plattformen portiert werden.

Acknowledgements

At this point, I would like to express my gratitude to all those who made this thesis possible. First of all, I would like to thank my advisor Prof. Dr. Wolfgang Effelsberg, who gave me the opportunity to join the "Lehrstuhl für Praktische Informatik IV" at the University of Mannheim and who guided me through my thesis. He has always been of great help and supported me with everything I needed. Working at his chair was a pleasure, since he gave me the freedom to find my own research interests and advised me in every possible way. I would also like to thank Prof. Dr. Colin Atkinson, who was so kind to co-examine this dissertation, and Betty Haire Weyerer, who spent so many hours on proof-reading. I am deeply grateful for her generous support.

Many thanks also go to my colleagues Holger Füßler, Benjamin Guthier, Thomas Haenselmann, Thomas King, Stephan Kopf, Fleming Lampi, Hendrik Lemelson, Sascha Schnaufer, Moritz Steiner, Tonio Triebel, and Matthias Transier. They were often a source of new ideas and their discussions and feedback helped me very much. Particularly, I would like to thank Thomas Haenselmann, who has always been generous with his time and advised me in many ways. I very much enjoyed the collaboration with him, our research discussions on Reed-Solomon and fountain codes, on algorithms for energy efficiency in common, and on the ideas for establishing connected topologies. His feedback was always of great help, and it gave me new ideas to think about.

Besides my research colleagues, I would also like to thank our system administrator Walter Müller, who helped me to overcome any kind of technical problems, and without him, I would have often been alone in the early hours of the morning. Last, but not least, I would like to thank our secretary Ursula Eckle, and Sabine Baumann, who handled all the University's bureaucracy issues for me.

Special thanks also go to Jürgen Vogel, who inspired me in many ways. During my studies, I worked with him as his student assistant, and he provided me with deep insights into the world of science. After I joined our research group at the University of Mannheim, he was still one of my biggest inspirations, and I very much enjoyed the time we worked together.

Above all, I am deeply grateful to my family. I would like to thank my sister Yvonne and in particular my parents Gudrun and Peter. I am forever grateful and indebted to them for their love and care. From the very beginning, they have provided me with support and guidance on my way that has now led me to where I stand today. I would also like to express my gratitude to my parents-in-law, Maria and Horst, who have always given me a second home.

Finally and most importantly, my deepest and most sincere thanks belong to my beloved wife Katharina. I am, and will always be, grateful to you for your love, your encouragement, and for being the joy of my life. Since we have met, you enrich my life, and you make me perfectly happy – I do not want to live without you anymore. You have ever believed in me, and it is you who brings out the best in me.

Contents

List of Figures

List of Tables

List of Abbreviations

ARQ	Automatic Repeat Request
ASCENT	Adaptive Self-Configuring Sensor Networks Topologies
ASK	Amplitude-Shift Keyed
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DARPA	Defense Advanced Research Projects Agency
DEC	Double Error Correction
DED	Double Error Detection
DSDV	Destination-Sequenced Distance-Vector Routing
DSN	Distributed Sensor Network
DSR	Dynamic Source Routing
EEPROM	Electrically Erasable Programmable Read-Only Memory
EGATE	Embedded Gate
ESB	Embedded Sensor Board
FEC	Forward Error Correction
GAF	Geographic Adaptive Fidelity
GBR	Gradient-Based Routing
GIST	Group-Independent Spanning Tree
GRAB	Gradient Broadcast
JTAG	Joint Test Action Group
LAN	Local Area Network
LP	Linear Program
LPT	Lifetime-Preserving Tree
LT	Luby Transformation
MAC	Medium Access Control
MEEAF	Multi-Link Energy-Efficient Aggregation Forwarding
MEEF	Multi-Link Energy-Efficient Forwarding
MEMS	Micro-Electro-Mechanical System
MIP	Mixed-Integer Program

MLEF ..... Multi-Link Lifetime-Efficient Forwarding
MSB ..... Modular Sensor Board
MST ..... Minimum Spanning Tree
MT ..... Minimum Transmissions
NRZ ..... Non-Return to Zero
OOK ..... On-Off Keyed
PRR ..... Packet Reception Ratio
RAM ..... Random Access Memory
RBP ..... Robust Broadcast Propagation
RF ..... Radio Frequency
RFM ..... RF Monolithics
RLF ..... Random Linear Fountain
ROM ..... Read-Only Memory
RS ..... Reed-Solomon
RT ..... Raptor
SEC ..... Single Error Correction
SEEAF ..... Single-Link Energy-Efficient Aggregation Forwarding
SEEF ..... Single-Link Energy-Efficient Forwarding
SLEF ..... Single-Link Lifetime-Efficient Forwarding
SMT ..... Steiner Minimum Tree
TECA ..... Topology and Energy Control Algorithm
TED ..... Triple Error Detection
TI ..... Texas Instruments
UART ..... Universal Asynchronous Receiver/Transmitter
WSN ..... Wireless Sensor Network

CHAPTER 1

Introduction

"We are what we repeatedly do. Excellence, then, is not an act, but a habit."

– Aristotle –

1.1 Motivation

Recent advances in the microsensor and semiconductor technology have opened a new field within computer science. In 1999, *Business Week* proclaimed the networking of microsensors as one of the key technologies for the 21st century [7]. According to Moore's Law [178], the number of transistors on a chip doubles at least every two years. At the same time, the processing power and storage capabilities grow. While this trend will continue, the electronic miniaturization and the advances in the semiconductor manufacturing process enable low-power and low-cost hardware. Small and smart devices equipped with a processing unit, storage capacity, and small radios offering wireless communication provide new application opportunities. Augmented with different kinds of sensors for, e. g., temperature, pressure, and humidity measurements as well as noise and movement detection, physical phenomena can be observed by deploying such sensor devices in their vicinity. The inexpensive production costs might allow for dense deployments in the physical space. For example, consider a scenario where thousands of devices, so-called *sensor nodes*, are dropped from an aircraft into a remote terrain [84]. At first, the sensor nodes must coordinate each other and establish a wireless communication network. Then, sensed phenomena can be processed locally, communicated to other nodes, and forwarded through the network. In general, the information will be forwarded to one or to several information sinks, which are special nodes connected to other networks. The sink nodes work as gateways and allow other applications to control the *wireless sensor network* (WSN).

Although the requirements of a WSN mainly depend on the application being used, we can characterize it by the following definition: *A WSN consists of a potentially large set of devices that are capable of sensing, processing, and communicating physical phenomena in order to meet a common application task by some kind of cooperation*. Even if an individual node is limited in terms of processing and storage capacity, complex phenomena can be observed and analyzed. The cooperation of

nodes is therefore important and one of the challenges a WSN must take into account. Since sensing and processing is distributed inherently, we can view the whole sensor network as a *distributed system* [58, 148]. By using only local information, each node contributes to the common application task and cooperates with other nodes in its neighborhood. Several protocols are needed to achieve such a cooperation [130]. Examples are protocols for the *medium access control* (MAC), routing protocols, or synchronization and localization protocols. However, it is unlikely that any WSN will use the same protocol stack as it is known from the Internet. It will rather depend on the application being used, since the application tasks as well as the deployments may be very different. Thus, the application scenario will determine the optimal choice of protocols.

The dense deployment of hundreds or even thousands of sensor nodes offers a wide range of new applications. Current and future application areas include habitat and environment monitoring, disaster control and operation, military and intelligence application, object tracking, video surveillance, traffic control, industrial surveillance and automation, as well as health care and home automation. Thus in the future, environment sensing will become more and more ubiquitous and part of our life. For example, modern vehicles already include several sensors connected to electronic systems in order to improve comfort and safety. Although those sensors are normally wired within a vehicle, they can also form a WSN if each vehicle is considered as a sensor on a larger scale.

Realizing such different kinds of WSNs is a great challenge, from an engineering as well as from a research perspective. Since the WSN as a whole serves a common application task, an individual node becomes less important. Thus, even if some nodes fail, the application task must still be feasible. Because of a possibly unattended deployment, the network must therefore adapt to changes in the environment and re-organize assigned sensor tasks. This behavior is referred to as *self-organizing* [46]. Self-organization is not only desirable when changes within the environment occur, but also as a means to disseminate data through the network. For example, pre-configuring each node with static routing tables would not be advisable since the reachability of nodes may change over time. It would also be very impractical to configure a large number of nodes manually. Designing self-organizing algorithms is not straightforward but must take the distributed nature of sensor networks into account.

Another key challenge a WSN has to deal with is energy efficiency because most sensor nodes may be battery-powered. In most cases, it may not be possible to change or recharge batteries, either due to the low-cost hardware being used or due to an inaccessible area the nodes are deployed in. To prolong the overall network operational lifetime, the energy consumption of a sensor node should thus be minimized as far as possible. Most of the nodes' components will therefore be turned off most of the time and will only be used if they are required. For example, the processing unit can be put into a low-power sleep mode while it is idle. Also, turning off the wireless communication radio conserves much energy since transmitting one bit consumes as much energy as about 1,000 processing instructions [67]. Thus, communication in a WSN is one of the main energy consumers and deserves particular consideration.

In order to fulfill these requirements, the protocols and algorithms used should be energy-efficient. Otherwise, an early failure of nodes due to lack of energy might require a reconfiguration or even cause significant malfunctioning within the network. Researchers are therefore currently focusing on power-aware protocols for different network levels [11].

At the *physical layer*, information intended to be sent to other nodes must be modulated onto the radio's frequency. The influence of disruptions on the channel can be minimized if *forward error correction* (FEC) codes are used. Bit errors within a received packet can then be corrected, preventing a retransmission of the entire packet by the sender. However, concerning energy-efficiency, there is a trade-off between FEC and *automatic repeat request* (ARQ), since error correction requires redundancy, which must be added to a packet. If the content contains no error after the transmission, the redundancy included will be useless, causing a needless energy consumption at the sending as well as at the receiving side.

The *data link layer* is also responsible for energy-efficient communication between nodes within transmission range. Due to the *shared medium*, packets from different senders may collide if they are transmitted at the same time. The *medium access control* MAC layer tries to minimize such collisions and schedules data transmission and sleeping periods. During the sleeping period, the communication radio of a node is turned off completely, which saves a significant amount of energy.

The *network layer* takes care of routing data through the network[1]. Therefore, forwarding paths are established. By exchanging the appropriate information, all nodes are able to discover other nodes in their neighborhood and learn about forwarding paths already established by their neighbors. In general, the communication will then take place along a tree rooted at a sink node. This will be the most likely communication scenario, as either sensed data will be sent towards the sink or the sink will query sensor nodes for information.

Finally, the *transport layer* takes care of the end-to-end data flow, taking into account the network congestion, and providing a reliable transmission service for the application layer.

Hence, any protocol used in a WSN should be power-aware and designed for energy efficiency. In contrast to traditional networks, the forwarding of data packet is not *address-based* but *data-centric*. That is, sensor nodes are not addressed by a globally unique identifier but rather addressed based on data attributes. For example, instead of requesting the temperature value of an individual node, an application may be more interested in whether or not there are any nodes which detect a temperature above a given threshold. Thus, this paradigm shift implies that a WSN will likely be tailored to sensing and application tasks. Furthermore, it enables the exploitation of application information during the forwarding process. *Application-specific* data forwarding may significantly reduce the amount of information that needs to be transmitted. By means of in-network processing and data aggregation, redundant and useless data can be filtered out along the forwarding path. For example, consider a scenario where a sink node is interested in the average temperature each node measures over a certain period of time. Rather than forwarding individual data readings, these can be aggregated by intermediate nodes along the path without loss by transmitting only the sum and the number of readings.

In conclusion, we can summarize the key requirements of a WSN as follows:

- Scalability: A WSN can consist of thousands of sensor nodes, densely deployed in a regional area. Protocols must scale well with such a number of nodes. This is often achieved by using distributed and localized algorithms, where sensor nodes must communicate with nodes in their

[1] We will refer to routing also as packet forwarding.

neighborhood only. Centralized approaches are often not applicable due to single points of failure.

- Adaptiveness: All protocols must be able to adapt to changes in the environment, e. g., connectivity changes or changes concerning the sensing of physical phenomena.
- Resistance to failures: Due to the low-cost hardware or outside influences, sensor nodes are prone to failure. However, this need not affect the achievement of the common application task. In case of node failures, the network must be able to re-organize itself and, if needed, change assigned application tasks.
- Self-organization: Since a WSN is usually deployed in an unattended area, the network must operate without any need of manual configuration. For example, communication paths throughout the network should be established automatically. Also the cooperation between nodes must be organized in an unattended manner.
- Energy efficiency: As most sensor nodes will have restricted energy capacities, all protocols and algorithms must be energy-efficient and save as much energy as possible. Since wireless communication consumes the most energy, the radio should be turned off most of the time. Also, the actual transmission of data must be energy-efficient, minimizing the number of packets sent and received.
- Simplicity: Finally, because sensor nodes are also limited as to their processing and storage capabilities, algorithms and protocols should be as simple as possible, reducing the computational complexity and memory usage.

1.2 Outline

The main purpose of this thesis is the design and development of energy-efficient protocols and algorithms for WSNs. As the radio communication consumes the most energy, we are looking for algorithms which minimize the number of transmissions throughout the network. However, we must always take into account the trade-off between the number of packet transmissions and the appropriate delivery rate in order to fulfill assigned application tasks. Otherwise, using no radio communication at all would be the most energy-efficient way.

The thesis is structured as follows: In the next chapter, we present the hardware platform we will use to demonstrate and evaluate the performance of our proposed algorithms. The platform was developed by the Free University of Berlin and mainly consists of *embedded sensor boards* (ESB). Based on an embedded chip from Texas Instruments, the MSP430, the ESB nodes were particularly designed with energy efficiency in mind. The boards are equipped with several sensors for temperature, light, movement, infrared, and vibration measurements. For wireless communication, an RFM TR1001 radio transceiver is used that allows for data rates of up to 115.2 kbps.

The radio transceiver of an ESB node is connected to a *universal asynchronous receiver/transmitter* (UART), which in turn is connected to the MSP430 chip. While this design allows a node to send and

receive complete byte streams (rather than single bits), it may also cause special transmission errors which we describe in Chapter 3. We will devise a *periodic resync scheme* that enables a receiver to catch up on the data stream sent by the sender and to re-synchronize itself with the sender's state machine in case bits are lost. Our resync scheme reduces the number of bit errors significantly, offering an efficient use of *forward error correction* (FEC). While FEC can be used to correct errors within a packet, it is also suitable for protecting an entire sequence of packets. For this purpose, we will consider a special class of FEC codes called *fountain* codes, that were recently proposed by other researchers, and compare their performance in a single-hop as well as in a multi-hop sensor network.

After considering the transmission between adjacent nodes, Chapter 4 focuses on the end-to-end packet delivery process between a sensor node and a fixed sink in the network. Because usually the network is large in size, each node is not able to communicate with the sink directly. Instead, data packets are forwarded by intermediate nodes and sent hop-by-hop along a forwarding path, which is based on the *reverse tree* established earlier by the sink. By using a realistic link loss model, we will consider different forwarding strategies and will propose two new schemes named *single-link* and *multi-link energy-efficient forwarding* (SEEF and MEEF). Both strategies optimize the forwarding paths in terms of their energy efficiency and trade off end-to-end delivery ratios and energy costs. While the SEEF strategy sends a packet to only one neighbor for forwarding, MEEF exploits the broadcast characteristics of the wireless medium by sending a packet to several nodes at once, from among which a forwarding node is selected afterwards. Through mathematical analyses, simulations, and an experimental evaluation, we contrast the performance of our approach against a comprehensive framework of other forwarding strategies proposed in the literature.

Although the forwarding paths established by SEEF and MEEF are optimized with regard to their energy efficiencies, the lifetime of the network is not maximized implicitly. Moreover, nodes located on optimal forwarding paths are used more often than others and thus consume more energy, thereby increasing the probability of network partitions if they run out of energy. In Chapter 5, we thus extend the SEEF and MEEF strategies by a lifetime component, which takes the residual energies of nodes into account. By incorporating the end-to-end delivery ratio, the required energy costs, and additionally the residual energy available on forwarding paths, we propose *single-link* and *multi-link lifetime-efficient forwarding* (SLEF and MLEF). As for SEEF and MEEF, we will present mathematical analyses, simulations, and results obtained from real-world experiments.

Chapter 6 describes another extension that accounts for data aggregation. Usually, physical phenomena sensed by multiple nearby sensor nodes are somehow correlated. For example, consider again a sensor network designed to capture the temperature in a predefined region. It is likely that most nodes in the immediate vicinity will sense the same temperature. Reporting all these values would be highly redundant and energy consuming, even if energy-efficient paths are used. On the other hand, the energy costs may be reduced considerably by in-network processing, which would increase the energy efficiency of the entire network. However, such cost reductions are not taken into account by SEEF and MEEF implicitly. The extension presented in Chapter 6, called *single-link* and *multi-link energy-efficient aggregation forwarding* (SEEAF and MEEAF), considers energy savings due to aggregation *a priori* and outperforms SEEF and MEEF, as well as other related forwarding strategies. Again, performance is compared by means of simulations and experiments.

While the algorithms presented in Chapters 4 to 6 focus on an energy-efficient forwarding process, they do not take real advantage of densely populated networks. Since commonly, many nodes are likely to be redundant, this feature can be exploited in order to recover from node failures or to prolong the overall lifetime of the network. Energy could be saved if redundant nodes switched to a low-power mode with their communication radios turned off, scheduling wake-up times periodically, or shortly before other nodes run out of energy. Establishing such a backbone topology of active nodes is part of Chapter 7, which presents an appropriate *topology and energy control algorithm* called TECA. TECA is designed particularly with regard to network connectivity, energy consumption, network lifetime, and load balancing. It comprises two parts, a *cluster head selection* and a *bridge selection* process. During the cluster head selection, the network is divided into adjacent clusters. Afterwards, the bridge selection process connects the different cluster heads, forming a topology of only active nodes. Using the concept of *virtual links*, passive nodes decide to join the backbone topology if necessary and become active in a distributed fashion. Other nodes are considered redundant and go into a low-power sleep mode. In contrast to three other considered approaches, TECA guarantees that the backbone topology is always connected. Furthermore, it trades off energy consumption and packet delivery ratios very well, achieving a high level of energy efficiency and a long network lifetime. All topology algorithms have been fully implemented and evaluated both in simulations and experiments.

As the last chapter, Chapter 8 provides a survey of sensor network applications and describes research projects that gained initial experiences with real-world networks. The chapter covers applications from habitat and environmental monitoring, health care, home automation, and the military. Even though the requirements of these applications differ, energy efficiency is always one of the most important common issues. Hence, the algorithms considered in this thesis may be of great value for future deployments.

Chapter 9 summarizes the thesis and presents final conclusions. It also emphasize our scientific contributions and gives an outlook on further extensions, as well as on possibilities for future work.

CHAPTER 2

The ScatterWeb Platform

"We shall do a much better programming job, provided we approach the task with a full appreciation of its tremendous difficulty, provided that we respect the intrinsic limitations of the human mind and approach the task as very humble programmers"

– A. Turing –

2.1 Introduction

ScatterWeb [5] is one of the earliest research projects in Germany that started to develop a platform for self-organizing wireless sensor networks. Founded by the Free University of Berlin a few years ago, ScatterWeb components were intended to be used for education and prototyping. Since early 2005, a spin-off company sells all components and additionally provides solutions tailored to industry-ready hard- and software. Today, several institutions in Europe are using the platform for research as well as industrial applications.

Since the ScatterWeb platform was one of the leading products about three years ago when our research lab was looking for appropriate hardware components, we decided to buy a comprehensive set of 30 sensor nodes and four gateways. Any other platform on the market could also have been used, e. g., the mica mote platform from the University of California at Berkeley [65]. But there were some advantages the ScatterWeb platform provided, like an easy way to program (flashing) and debug, a low implementation overhead concerning protocols and applications due to the standard C language, and the fact that the platform already comprised several sensors on-board. Thus, in this thesis, we will use the ScatterWeb platform as an example to describe and demonstrate our algorithms. Nonetheless, other platforms are expected to show similar characteristics as they certainly encounter similar restrictions regarding processing, communication, and memory capabilities.

2.2 The Embedded Sensor Board

The actual sensor node of the ScatterWeb platform is the *embedded sensor board* (ESB), which is equipped with processing, sensing, and radio transmitting components. Figure 2.1 shows an image of an ESB node that has a size of 5×6 cm^2. Most of the space is required for the sensor hardware and the battery pack containing 3 AA Mignon batteries.

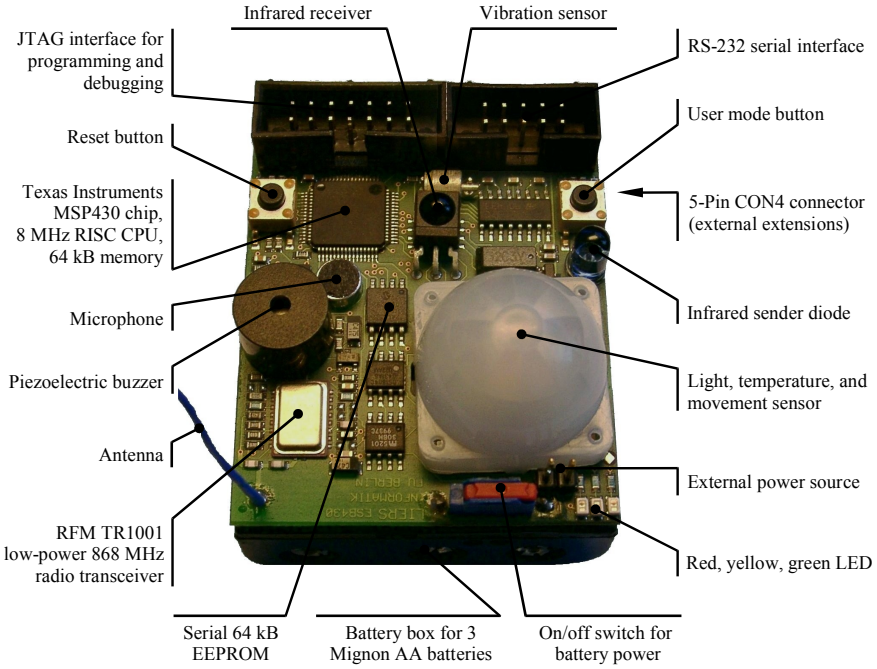


Figure 2.1: Embedded sensor board

The development of new applications and protocols is quite simple because the complete code is written in standard C. No additional programming language is required, which significantly shortens the time to set up a node. Over the *joint test action group* (JTAG) interface, which is depicted in the upper left corner, the node can then be flashed and debugged in a very convenient way. Connected to the parallel port of a PC or a notebook, the firmware as well as programmed applications are flashed into the memory at once. After a hardware reset, the firmware reboots and starts the application. Real-time debugging is also provided via the JTAG interface, offering common functions like breakpoints, single-step debugging, and variable monitoring. Another way to flash the node's memory is by *over-the-air* programming that is quite beneficial if several nodes need to be flashed at once. However, in order to work properly in such a case, the byte code of the firmware must not be changed.

Next to the JTAG interface, a standard RS-232 serial interface is provided that can be used to connect the ESB node with a terminal program running on a PC for input and output. The data rate is between 300 bps and 115.2 kbps (default). In this way, state or debug information can be sent to the terminal and commands can be transmitted to the node. Furthermore, the serial interface also allows the node to connect with wide-area networks via a mobile phone using GSM/UMTS. For example, a special command can be sent to the mobile phone in order to trigger the transmission of SMS messages. In

addition to the RS-232 interface, there exists a 5-pin CON4 connector (not displayed) for external extensions via synchronized serial communication in SPI mode.

2.2.1 MSP430 Microcontroller

The ESB node is equipped with the MSP430F149 [24], an embedded system on a chip from Texas Instruments, running at 8 MHz. Due to its low energy consumption and an appropriate power down mode, it is quite suitable for battery-driven applications. The currently used version of the MSP430 contains 64 kB memory that is almost completely implemented as flash memory (ROM). The RAM available to the firmware and application is only 2 kB, limiting the implementation possibilities of new protocols considerably. Despite the fact that it is feasible to write 128 byte chunks into the flash memory during operation, dynamic data is difficult to maintain. While reading from either flash or normal memory makes no difference, writing into the flash memory is both time- and energy-consuming. The same applies to the external 64 kB EEPROM, an *electrically erasable programmable ROM*. As almost the entire EEPROM is for free use, it is suitable for storing large data blocks like routing tables or, e. g., images captured by a camera connected via the serial interface. In addition, the EEPROM is used for storing the node's configuration as it remains unaffected by flashing.

The internal memory of the MSP430 is organized according to a predefined memory map, as shown in Figure 2.2. Thus, depending on the address, code and data are handled differently. For example, after a block of 60 kB that is used for the firmware and application, there are two blocks of 128 bytes allowing flashing from within the program. Furthermore, some memory is reserved for a boot-loader which handles the actual re-programming. During flashing, the program counter is restricted to this address region so as not to cause a conflict with other code being processed. Thus, as long as the boot-loader is in operation it must not be overwritten.

Addresses	Memory block	Description
0xFFE0 – 0xFFFF	Interrupt vector table	16 sub-routine addresses
0x1100 – 0xFFDF	Flash-ROM for firmware, programs, data, and tables	Ca. 60 kB flash memory, written at once during programming over the JTAG interface
0x1000 – 0x10FF	Information memory A and B	2 x 128 bytes memory block
0x0A00 – 0x0FFF	Boot-loader ROM (fix)	Programmed over JTAG
0x0200 – 0x09FF	RAM for variables and stack	2 kB RAM
0x0100 – 0x01FF	8-bit periphery	Memory-mapped
0x0000 – 0x00FF	16-bit periphery	Memory-mapped

Figure 2.2: Memory map of the MSP430

To connect different types of sensors, the MSP430 provides an internal 12-bit A/D converter, transforming between analog and digital signals. The A/D convertor provides eight external and four internal inputs and has a conversion time of less than 10 μs. The conversion requires no interrupt handling, using 16 8-bit registers to read and 16 12-bit registers to store the results. In this way, sampled values from an external sensor can be read like data from memory. In addition, the memory is used to *control* external sensors by writing into special addresses and registers. This technique is commonly known as *memory mapping*, which links the outside world with the embedded system. For example, by writing into so-called *memory mapped ports*, sensors can be configured and powered down. Moreover, memory mapping can be used to integrate external processors. Although the CPU of the MSP430 does not provide an internal integer multiplier, multiplication of two integers can be performed by an external unit: a hardware multiplier is connected with the CPU by mapping the multiplier's registers into the memory of the MSP430. Upon writing both operands into memory, the multiplication is started automatically, returning a 32-bit value at a predefined address.

In addition to a basic timer used by the CPU, the MSP430 offers two additional timers based on 16-bit counters, which can be used freely, e. g., for serial or wireless communication. Both timers are able to operate in three different modes and with different clock sources, depending on the precision the timers are intended for. Another feature is provided by a watchdog that comprises a timer and a control register. Starting at a predefined value, the timer decreases automatically, resetting the processer as soon as the timer expires. Usually, the timer is reset periodically by the main loop of the operating system. However, if the firmware or the program crashes or is locked somehow, the watchdog will trigger a re-initialization of the system.

2.2.2 TR1001 Radio Transceiver

The radio communication of the ESB platform is based on the RF Monolithics TR1001 radio transceiver [200], which has a very small footprint, as depicted in the lower left of Figure 2.1. Furthermore, it requires very little power, keeping the energy consumption of the whole system low. Otherwise, the TR1001 is a very simple transmitter, providing both *on-off keyed* (OOK) and *amplitude-shift keyed* (ASK) modulation. The radio range depends heavily on the environment. It can range from several hundred meters outdoors to less than ten meters indoors. Using OOK modulation, data transmission rates of up to 19.2 kbps are possible; ASK modulation even allows up to 115.2 kbps. However, practical experiments have shown that the bit error ratio increases dramatically for data rates above 38.4 kbps.

The transceiver is able to monitor the receive power and to adjust its transmission power. The maximum transmission power is about 1 mW, while the amplitude of a signal is proportional to the current at the input transmission pin. The current can be controlled by software, scaling the transmission power to values between zero and one hundred percent. Besides basic functions like bit modulation and sampling, the radio transceiver does not perform any other tasks. Thus, additional components as well as higher-layer protocols are required in order to transmit a byte stream conveniently.

Sending a packet, as provided by the firmware of the ESB platform, is done as follows: By means of a *universal asynchronous receiver/transmitter* (UART) that is connected with the radio transceiver, a

byte stream is first serialized and then transmitted by the radio transceiver. Writing single bytes into the UART buffer is done periodically, depending on the transmission rate used for the radio. Triggered interrupts inform the firmware when the next byte needs to be written into the buffer and when the transmission over the radio is completed. To send an entire packet, all bytes of the packet are first queued into a ring buffer, which is afterwards used to feed the UART buffer. The interrupt handler is then responsible for the actual transmission, including random backoff times to avoid collisions, CRC checksums, and automatic retransmissions.

The receiving of a packet works in a similar way. Received bytes are de-serialized by the UART and read from the UART buffer by means of another periodically triggered interrupt. Again, a ring buffer is used to hold received packets in order to provide access to them for the application. Depending on the type of a packet, i. e., whether or not it is a unicast packet, an acknowledgement is generated and sent back to the sender automatically. Duplicate packets are recognized by means of sequence numbers.

2.2.3 Sensors and Equipment

As illustrated in Figure 2.1, the ESB platform is also equipped with the following components:

Passive Infrared Sensor Hidden under the white fresnel lens, a passive infrared sensor is located, allowing for monitoring the space around an ESB node. The detection range is approximately eight meters. Thus, the sensor is useful for motion sensing or intrusion detection.

Vibration Sensor In order to reliably detect moving objects, a vibration sensor is also provided, which is able to detect vibration ranging from nearby moving persons to earthquakes.

Light Sensor Besides the passive infrared sensor under the fresnel lens, a light sensor is installed that allows for precise measurements of light intensities. Furthermore, the sensor can differentiate between artificial and natural light.

Temperature Sensor Integrated into one chip, a digital thermometer and a real-time clock are provided. The accuracy of the thermometer is about ± 2 C, using a resolution of 9 bits. The clock provides time and data information. Thus, it is possible to trigger an alarm that can be used as an input for the microcontroller. For example, the alarm output is activated if either the measured temperature exceeds a programmed temperature limit or the current time reaches a programmed alarm setting.

Infrared Receiver and Transmitter Infrared communication is provided via appropriate diodes for sending and receiving infrared signals. In this way, the ESB node can be controlled by using standard RC-5 codes such as those used by remote control devices of consumer electronics. Furthermore, the node itself can send commands to any infrared-enabled device.

Microphone Using the microphone, noise detection can easily be implemented. As for all sensors, the readings can be used to wake up the MSP430 controller, using a special interrupt. Thus, an application does not have to check for appropriate conditions itself.

User Mode Button In addition to a reset button which resets the system, a user button is available that is freely programmable. E. g., upon pressing the button, the application may be started and stopped, or user-defined events may be triggered.

Piezoelectric Buzzer There is also a piezoelectric buzzer placed on the board in order to interact with the physical world, e. g., to signal errors or emergency situations. As the buzzer is quite loud, even long distances can be bridged.

Red, Green, and Yellow LEDs And last, the ESB platform has three LEDs (red, green, and yellow), which are quite helpful to signal state information during operation. In particular, the LEDs are useful for debug purposes.

2.2.4 Power Consumption

The power consumption of the ESB platform running with all sensors is around 12 mA. Transmitting data causes an additional consumption of about 8 mA, depending on the transmission power being used. In a low-power mode (deep sleep) where all sensors and the ratio transceiver are turned off, the consumption is approximately 8 μA only, which is about 1,000 times less compared to its normal operation mode. Equipped with the 3 AA battery holder, the lifetime of a node would be about 5 years, assuming a duty cycle of 1% and neglecting the self-discharge of the batteries[1]. The lifetime can even be extended if the node is equipped with an external solar panel which allows for recharging. Variations of the input voltage are stabilized by a special controller to 3 V. Hence, a higher voltage will cause no damage. Furthermore, the voltage is permanently monitored, triggering an event if the input voltage falls below a given threshold.

2.3 The Embedded Gate/USB

A more convenient way to access one or several ESB nodes is provided by the *embedded gate/USB* (eGate/USB) pictured in Figure 2.3. The board basically consists of the same main components as the ESB, but without any sensing hardware. Also, no RS-232 serial interface is provided. Instead, the eGate/USB acts like a USB stick and is installed like common USB devices on a PC or even a PDA. Technically, a special converting chip is used to perform the conversion between serial access and USB. Thus, connected to a terminal program, the eGate/USB can be accessed like other ESB node using the RS-232 interface. An external power supply is not necessary because the power provided by the USB port can be used.

The MSP430 on the eGate/USB can be flashed via the parallel port of a PC, too. This requires that a particular adapter connects the 5-pin JTAG connector (shown in the lower right of Figure 2.3) to a common parallel cable.

[1]The duty cycle of a node specifies the fraction of time in which the device operates. A duty cycle of 1% thus means that the node, e. g., runs for 1 second and then sleeps for 99 seconds.

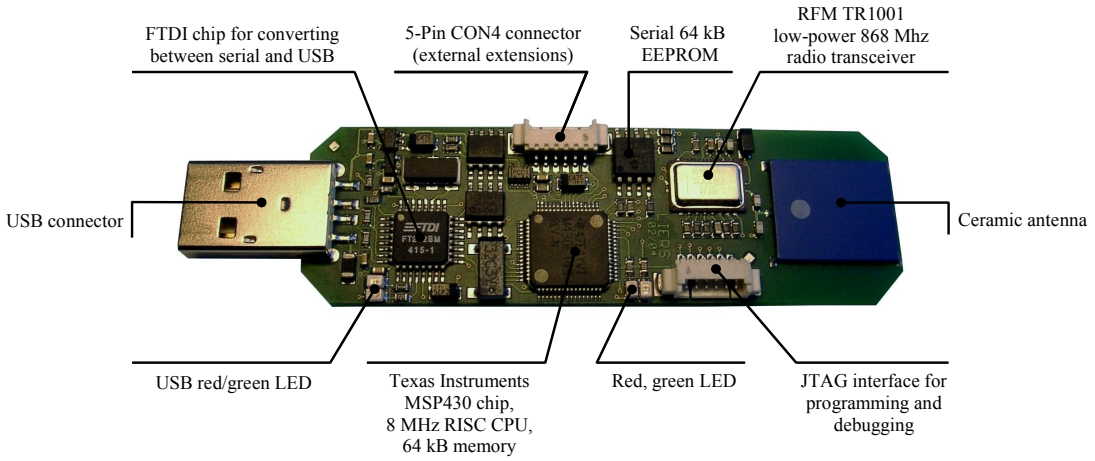


Figure 2.3: Embedded gate/USB

To enable communication with other devices, the eGate/USB is equipped with a wide-band ceramic antenna which has a footprint of only 2 cm^2. The actual radio communication uses the same TR1001 transceiver as described above. Flashing several nodes over the air at once is then carried out as follows: Via the USB connection an image containing the firmware and an application is transferred to the eGate/USB and written into the EEPROM. Afterwards, the image can be sent over the RF interface to one or to several nodes within transmission range. The image is divided into multiple chunks. Again, upon receiving, the image is stored into the EEPROM for processing. Note that due to the broadcast characteristics of the wireless medium, the image needs to be transmitted only once in the majority of cases. Only if chunks get lost, they will be retransmitted on request. If an image is received correctly and is additionally consistent with the boot-loader being used, the memory is flashed with the new code. Finally, the control register of the watchdog is used to reset the embedded system.

2.4 The Embedded Gate/WEB

Another way to gain access to a network of sensor nodes is offered by the *embedded gate/WEB* (eGate/WEB) device. The eGate/WEB differs from the eGate/USB mainly in the connection of the board. Instead of using a USB connector, the eGate/WEB uses the Digi Connect ME system, an embedded module that provides for web-enabled network connectivity (located on the left in Figure 2.4).

Connected over Ethernet, the Digi Connect ME module can be accessed and set up using HTTP. Properly configured, it then allows a telnet client to access the eGate/WEB via TCP/IP, converting between Ethernet and the serial interface of the MSP430. Compared to the possibilities of the eGate/USB, this is particularly convenient if the controlling PC is far away. Furthermore, connected to a WLAN router, the network can easily be augmented by PDAs, featuring a special kind of user interaction.

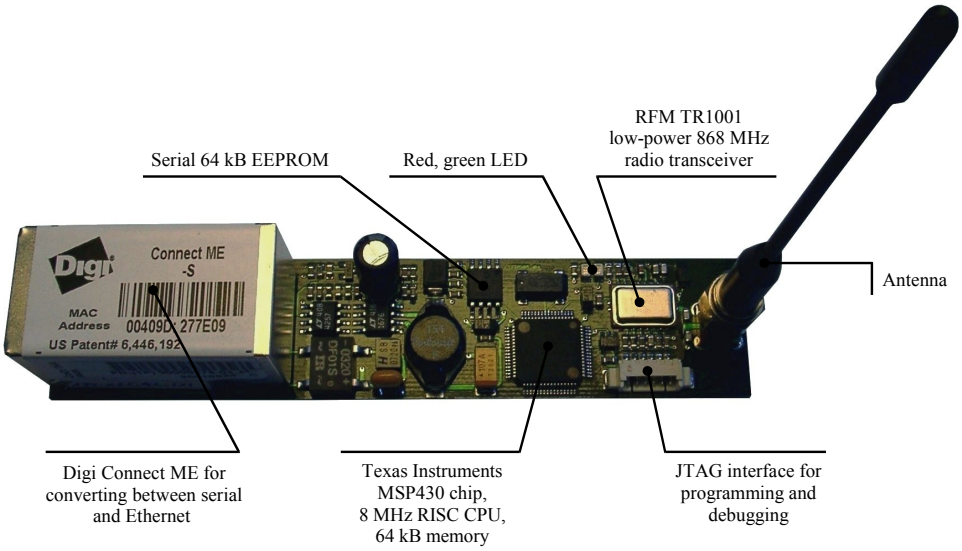


Figure 2.4: Embedded gate/WEB

2.5 Conclusions

The ScatterWeb platform consisting of the ESB, eGate/USB, and eGate/WEB boards offers a flexible and convenient way to develop and install a wireless sensor network. Although they are yet far from being densely deployed as it was envisioned by the *smart dust* project [245], the ESB nodes perfectly matches the requirements of this thesis.

In the following, we use the ScatterWeb platform to demonstrate all algorithms developed in this thesis. Besides a gain in practical experiences, this provides us with a first proof of concept that each algorithm can really be implemented on such resource-restricted hardware. However, using the radio transceiver as an example, the next chapter also shows the shortcomings of such an embedded system.

CHAPTER 3

The Impact of Resync and Forward Error Correction

"Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."

– D. Knuth –

3.1 Introduction

In this chapter, we describe a resync mechanism that allows two sensor nodes to re-synchronize their communication after a transmission error has occurred. During experimental measurements we have observed that bit errors are not evenly distributed over a received packet. Our resync mechanism can significantly reduce the effect of such an uneven error distribution, which is due to special transmission errors. These errors are caused by some commonly used hardware components, particularly the so-called *universal asynchronous receiver/transmitter* (UART) circuit (also used in serial communication) that interconnects the radio transceiver and the CPU. Because of the asynchronous communication between two nodes, the UART generates a start and stop bit in order to frame the actual data bits. As long as the state machine at the sender is synchronized to that at the receiver, no resync is necessary. However, once one or several bits are missed, the state machine at the receiver side will get out of sync so that data bits are misinterpreted as start or stop bits and vice versa, rendering the remaining communication useless. But even in the case of skipped bits, the proposed resync mechanism enables the receiver to catch up on a data stream, thus substantially reducing the number of errors in a corrupted packet. Remaining errors can then be corrected by means of *forward error correction* (FEC) codes.

In the second part of this chapter, we will consider such FEC codes, consisting of a single-bit error correction code, a double-bit error correction code, and a Reed-Solomon code. We will also analyze

the impact of interleaving, which is a well-known technique to account for burst errors. As such burst errors are likely in wireless communications, especially if the sender and receiver are out of sync, interleaving may improve the efficiency of forward error correction under certain conditions.

In the last part, we will address the case of data dissemination in a multi-hop sensor network. Assuming an application scenario where a large amount of data is to be disseminated to every node in the network, we consider different approaches and analyze their performance by means of experimental evaluations. Again, one possibility is the usage of FEC, but unlike before by spreading the code over several packets. While Reed-Solomon codes are quite suitable for this purpose, we will also consider another class of FEC codes, the so-called fountain codes. The advantage of fountain codes is their independence regarding packet loss on the wireless radio channel. Particularly if data is broadcast to multiple receivers in a lossy environment, fountain codes outperform Reed-Solomon codes as well as non-coding significantly.

3.2 The Resync Mechanism

During first measurements concerning the transmission quality of the ESB nodes, we encountered poor delivery ratios for low transmission powers and large inter-node distances. We also encountered a high variation among the nodes and thus analyzed the error characteristics in more detail. Other researchers reported similar transmission errors they obtained with other hardware platforms [257, 276, 278, 282]. In particular, the variation of the loss rate among nodes seems to be caused by the low-cost hardware being used and by the low-priced manufacturing process.

Analyzing the occurrence and the number of bit errors contained in an erroneous packet shows that the probability of a bit getting inverted is not distributed evenly, but rather is a rising function rising monotonously with the length of the packet. The probability of a bit error thus seems to be dependent on the probability of errors that occurred before. That is, the longer the packet, the higher the probability of an inverted bit. Since it is unlikely that this behavior is caused by the radio channel, it must be related to the hardware itself. As we will describe in the next section, the high number of bit errors at the end of a packet is due to the use of UARTs, which are well-known from serial connections on PCs. The UART links the radio transceiver with the CPU and provides asynchronous communication. However, in some situations the UART misses single bits such that the following bits are shifted to the left with respect to the true bit stream being sent. Once the UART has lost a bit, it might start to misinterpret the boundaries of bytes, receiving random information as a consequence. Recovering these random bytes is then no longer possible. Intuitively, shifting them a few bits to the right would align them with the true stream again. But getting the stream re-synchronized is quite a challenge which we tackle in the following.

Although our resync mechanism is not able to prevent bit errors which may occur during the transmission of data packets, it reduces the number of errors per packet significantly. It thus allows for a reasonable employment of FEC codes. Without resync, FEC is clearly outperformed by *automatic repeat request* (ARQ) as reported in [257] since usually there are too many errors that need to be

corrected. The redundancy added by a FEC code may thus be useless because it may not be sufficient to correct all bit errors of a packet.

In the next section, we consider the communication characteristics of the ESB nodes and describe how the UART is involved in the communication process. We then present the ideas for our resync mechanism and evaluate it by using a comprehensive set of measurements.

3.2.1 Communication Characteristics of ESB Nodes

The ESB platform was mainly designed with regard to energy efficiency and the possibility to directly interact with the radio transceiver. Due to the latter, no dedicated radio controller as used in Bluetooth devices [4], cellular phones [172], or many IEEE 802.11 Wireless LAN cards [160] is employed. A dedicated controller reduces the CPU load significantly since it handles the internal communication with the radio transceiver itself, including channel encoding, bit synchronization, and the detecting of the transmission start. It also provides a high-level interface to the CPU, refining the raw data stream received from the radio. In spite of these advantages, the dedicated controller prevents a higher-layer application from controlling the radio directly and hides useful information about the communication. For example, a fine-grained control that is needed by many MAC protocols [81, 241, 271] is no longer available. Also, access to time information about packet transmissions and arrivals, which is used in time synchronization protocols [83, 89, 88] or by localization algorithms [99, 182, 230], is prevented by a dedicated controller.

Thus, only a single controller is used on the ESB platform, facilitating arbitrary applications to directly control the radio and to gather all information needed for further processing. However, in order to extract and drive the bits to and from the radio interface, a precise timing is necessary, which must be provided by the CPU itself. Since a synchronous communication is quite expensive in terms of energy, a UART is used to connect the CPU to the radio transceiver. Besides its asynchronous communication possibilities, an additional advantage of the UART is that it accepts entire bytes, which are then transmitted to the radio transceiver. Thus, by using periodically triggered interrupts for sending and receiving, the CPU is allowed to enter a low-power sleep mode or to perform other work during the serialization and de-serialization, respectively. However, as we will see, using a UART for wireless communication in lossy environments is not advisable.

Universal Asynchronous Receiver/Transmitter

We now consider in more detail how the UART drives the bits to the radio transceiver. A *universal asynchronous receiver/transmitter* (UART) is basically an 8-bit shift register used for serial communication [256], mapping data between serial and parallel interfaces. In embedded systems, UARTs are commonly used with other communication standards such as RS-232 [24]. At the sender side, the UART serializes a stream of bytes into a sequence of bits, which are reassigned to bytes at the receiver side. In order to use the UART in conjunction with radio communication, the input pin of the radio transceiver is fed with a sequence of bits from the output pin of the UART.

To support asynchronous communication, a byte is framed by a preceding start bit (logical 0) and a succeeding stop bit (logical 1), as shown in Figure 3.1. The start bit is represented as a low signal level (L), the stop bit as a high level (H). At the receiver, a low level is interpreted as a start bit whenever the UART state machine is idle. Thus, the problem is that in the worst case the receiver might skip one or more ones until it encounters a falling edge. The next zero bit is then misinterpreted as the start of a new byte frame. In such a case, the receiver will be out of sync for the entire remaining bit stream.

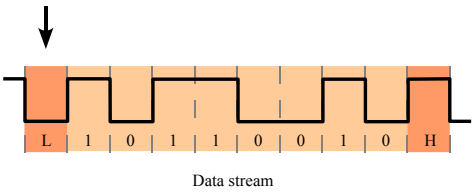


Figure 3.1: A byte stream framed by start (L) and stop (H) bits

To avoid an erroneous detection of start bits, three samples are taken around the middle of a bit. If a start bit is detected successfully, the receiver samples the following eight bits and waits for a stop bit. If no stop bit can be detected, a frame error occurs. Nevertheless, the received byte is transferred to the UART buffer and reported to the CPU for processing. Afterwards, the UART becomes idle again and waits for the next $1 \rightarrow 0$ transition.

The sender works in a complementary way. Whenever the UART is idle, the next byte waiting for transmission is transferred from the UART buffer to a shift register. Before the byte is serialized, the UART generates the start bit. The start bit, all data bits, and finally the stop bit are then transferred to the radio transceiver.

Data Transmission Characteristics

As mentioned above, at the sending side a logical 0 is mapped to a low voltage level and a logical 1 to a high one. At the receiver side, the received baseband signal is converted to a voltage that is linear to the strength of the signal. A voltage higher than a predefined threshold is interpreted by the radio transceiver as a logical 1, a voltage lower than the threshold as a logical 0. According to the current baseband voltages, the radio receiver will tune the threshold, i. e., the threshold might be increased or decreased. For optimal reception, the threshold should thus be tuned to the middle of the highest logical 1 and the lowest logical 0 level. Therefore, each byte must roughly contain the same amount of ones and zeros, which is referred to as being *DC-balanced* [201].

DC balance is achieved by appropriate encoding schemes, either used at the physical layer or at the link layer. Figure 3.2 depicts an example of a *non-return to zero* (NRZ) and a Manchester encoded data stream. Unlike the NRZ code, the Manchester code is DC-balanced since each bit is either encoded as a transition between low and high (logical 0) or between high and low (logical 1). However, note that compared to the NRZ code, the baud rate is only half the bit rate.

Since the TR1001 radio transceiver on the ESB platform only supports NRZ encoding, DC balance must be achieved by software. Therefore, a byte is encoded into two bytes as shown in Figure 3.3.

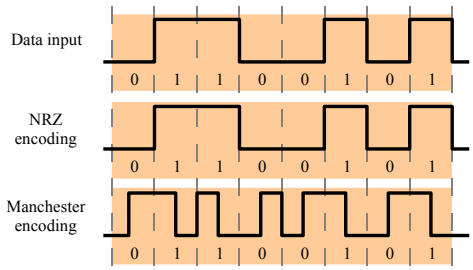


Figure 3.2: Example of NRZ and Manchester encoding

The first byte consists of the odd bits; the second byte contains the even bits. In order to fulfill the DC balance property, each bit is followed by the appropriate inverted bit. Note that this encoding scheme is not the same as the Manchester code, but can be implemented very efficiently. After a byte is encoded, both encoded bytes are transmitted to the UART, which in turn sends a serial stream to the radio transceiver.

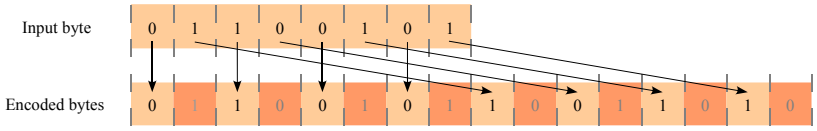


Figure 3.3: Encoding scheme used by the ESB firmware

To synchronize the receiver side to the beginning of a packet transmission, a *preamble* is sent first, followed by a synchronization byte, two start bytes, and the actual data. The preamble consists of five `0xAA` bytes (which is a sequence of alternating bits) in order to tune the receiver's radio transceiver. The following synchronization byte `0xFF` (idle line) synchronizes the receiver's UART to the start bit of the next byte. In order to help the receiver to recognize the start of the packet and to distinguish it from noise, two predefined start bytes (`0x01` and `0x7F`) are used[1]. The idle condition of the second start byte allows the receiver to recognize the start bit of the data block more reliably.

The entire packet structure is illustrated in Figure 3.4. The data block consists of an 8-byte packet header, followed by an arbitrary length of application data. The packet header is composed of the destination and source node's identifier, a packet type identifier, a sequence number, and the length of the contained data. To ensure a reliable communication, a 16-bit CRC checksum is additionally inserted at the end of the data stream.

Specific Error Characteristics

In several experiments with the ESB platform we have observed that the frequency of bytes being corrupted almost monotonously increases with the number of transmitted bytes. Since radio disturbances should actually be independent of the byte's position, we assumed that most of these errors are

[1]The start bytes must be distinguishable from other bytes. They therefore consist of two bytes containing a run of zeros followed by a run of ones to keep the transceiver DC-balanced.

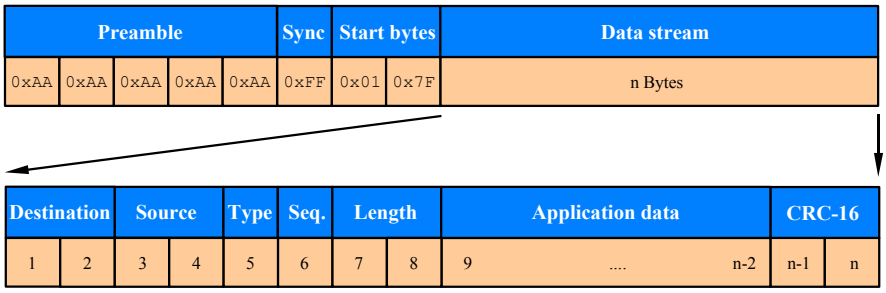


Figure 3.4: Entire packet structure consisting of a preamble, header, and data block

somehow caused by the hardware itself. We therefore captured some received packets and analyzed the raw data stream in more detail.

Figure 3.5 shows an example of such a received byte stream, depicted by hexadecimal numbers that illustrate the packet content. The data stream starts with the two start bytes `0x01` and `7F`, followed by a packet header and some test bytes. Note that due to the encoding scheme described in the previous section, each byte is encoded into two[2]. For example, the first two bytes of the actual packet (`0xAA` and `0x55`) are decoded to `0xFF`, indicating that the packet was sent per broadcast.

```
01 7F AA 55 AA 55 59 A6 55 AA 96 A6 A9 5A 55 AA 55 A9 AA 65
99 A6 6A 56 A9 59 95 A9 65 96 AA AA 59 6A 5A 5A 96 99 5A AA
A6 5A 56 69 66 6A 56 6A 96 95 AA A5 56 59 55 9A 99 A5 A6 66
A5 A9 9A A6 51 96 66 6A AA 69 A5 99 AA 66 69 59 56 6A 56 96
95 69 69 95 95 A9 AA 56 5A 55 A5 A9 99 56 96 95 95 5A 99 55
A9 96 69 55 9A 66 56 99 9A 69 95 AA 69 59 66 9A 5A 6A A9 AA
59 66 5A 99 AA A9 66 56 55 5A 56 AA 59 56 69 65 9A A5 56 95
A5 69 95 59 99 66 66 66 65 A5 AA A5 65 59 A9 56 56 9A 59 95
99 69 59 66 5A 55 96 69 6A 65 95 59 A9 59 9A 9A 56 A5 A5 56
96 66 A5 56 6A A9 6A A5 56 56 A9 95 A9 AA 66 59 9A 9A 55 AA
AA 59 4B 33 33 67 99 65 A9 99 99 59 9A 99 56 A9 56 96 64 9A
99 2B 2B 95 95 66 69 56 5A 69 65 AA 56 56 99 6A 9A A6 61 56
5A 66 66 5A A9 69 95 59 65 65 65 9A A6 45 69 69 65 4B 66 6A
59 99 AA A6 55 99 A5 99 AA 6A A5 A5 66 96 6A 6A 95 A6 96 69
56 59 A6 65 99 65 AA 4A 59 9A 69 A5 95 A5 69 69 A6 65 65 AA
99 66 65 5A 69 56 9A 99 A5 66 99 6A 16 59 95 6A AA A9 55 59
```

Figure 3.5: Part of an erroneously received packet

Erroneous bytes are labeled with a orange box, but there are two things that should be noted:

1. The byte `0x51` is wrong due to a single bit error since the correct byte was `0x59`.
2. With the next erroneous byte `0xAA`, all following bytes will either show bit errors (red text color) or will be misinterpreted (green text color). Misinterpretation means that a byte is received without an error but at the wrong time in the data stream.

[2]For the same reason, only the bit sequences 0101 (`0x5`), 0110 (`0x6`), 1001 (`0x9`), and 1010 (`0xA`) occur.

Due to the second observation, we assume that the receiver runs out of synchronization near the end of the packet. If we take a look at the disturbed bytes, we can recover the original byte stream. However, the byte stream (indicated by the green text color) is shifted to the left by a number of bytes. Unfortunately, it is not possible to recover the number of bytes the stream has shifted, unless the content of the packet is known *a priori*.

By considering the hardware involved in receiving data over the radio, we can explain these errors as follows: Single bit errors are actually rare and are not limited to data bits only; it is also possible that the start bit, which is used by the UART for the byte synchronization, will be disturbed. However, the loss of a start bit has serious consequences for the following bytes. In the first place, the receiver gets out of sync, and the following bits are shifted to the left. Secondly, some bits of the next byte might be *dropped* until the UART has received the next start bit. Hence, it is possible that a single-bit radio disturbance can corrupt an entire packet.

Figure 3.6 illustrates such a scenario, showing an original data stream framed by start (L) and stop (H) bits, and a corresponding stream in which the first start bit is corrupted (E). Considering the corrupted stream, the UART of the receiver skips all bits after the first stop bit until it detects the next logical 0, which will be interpreted as the start bit of the next byte. However, in this case, the start bit is already part of the detected data byte. Thus, the following eight bits are misinterpreted as data. At the end, the stream is shifted about three bits compared to the original one.

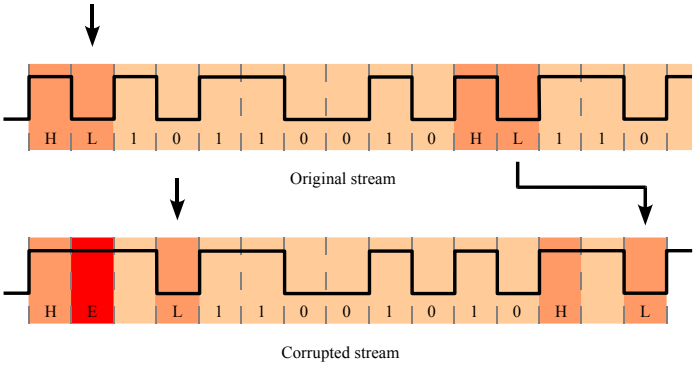


Figure 3.6: An original data stream and a corresponding corrupted stream

3.2.2 Design of an Appropriate Resync

As we have seen, most of the errors occurring are due to an unsynchronized UART that has lost one or more bits and thus misinterprets data bits as start or stop bits, or vice versa. This leads to an unrecoverable misinterpretation of all following bytes. Thus, the disturbance of start bits is most critical. Since the receiver does not know the transmitted data a priori, it will not be able to re-synchronize to the original data stream by itself. Rather, the sending node must provide a mechanism that allows for an automatic resync.

As discussed in the previous section, a UART performs byte synchronization by means of start and stop bits. Since a start bit will be detected by a falling edge, the best way to carry out a resync is to restart the transmission with an idle line, i. e., a consecutive run of ones.

Our solution becomes more intuitive if we regard both the sender and the receiver as state machines. As long as both sides are synchronized, a start bit at the sender side will be interpreted as such on the receiver side. The same is true for data and stop bits. But once the receiver has missed a bit somewhere, its state machine will fall behind. Consequently, it will read the next start bit as a data bit and, upon sampling eight bytes, the first falling edge in the original data section as a start bit. To resolve this situation, the sender sends eight consecutive high values followed by a falling edge, so that the state machine of the receiver will be in the state "*waiting for a falling edge*", no matter how many bits the receiver fell behind before. From that time on, both state machines will be in the state "*start bit detected*".

We can conclude that the UART's state machine is not very adaptive. It usually changes from a preceding state to a single succeeding state. Only when waiting for a start bit, does "*no falling edge*" mean "*stay in your current state*", while "*falling edge*" means "*start reading bit one*". This is the only opportunity at which we can make the receiver wait until both state machines are synchronous again.

In order to make the receiver's state machine wait occasionally, we stuff resync bytes (`0xFF`) into the data stream. `0xFF` maps to a long run of high signal values on the channel, which can be interpreted as an anchor if the receiving UART gets out of sync. This kind of anchor must occur in the data stream periodically because the sender cannot know when the receiver might be out of sync. That way, the receiver will be able to recognize the start of the following byte. No matter how many bits were skipped coincidentally, the UART will detect a falling edge after the sequence of 1-bits (`0xFF`) in any case.

To achieve a resync at the byte level, the following approach is used: Let n be the number of bytes after a resync byte is stuffed into the stream and m be the number of yet received (encoded) data bytes by the receiver. The stuffing starts with the first data byte (after the start bytes `0x01` and `0x7F`). Thus, the receiver always expects a resync byte after n bytes, i. e., if

$$m \mod n = 0. \tag{3.1}$$

For each byte received, the receiver then checks if it is a resync byte (`0xFF`). To be more robust against radio disturbances, 1-bit errors within a resync byte are ignored[3]. For example, `0xFB` is considered as a resync byte, too. If a resync byte is detected, the receiver will verify if the byte stream is still synchronous. Otherwise, Equation 3.1 is false and so is the receiver's byte position. Since it is likely that the receiver will be some bytes behind, it will increase its byte position m within the data stream until Equation 3.1 is satisfied. If no resync byte could be recognized but m has become a multiple of n, the receiver might have missed the resync byte. However, it is likely that the resync byte was already considered as a data byte. Hence, the receiver will skip the current byte and wait for the next resync.

According to Figure 3.6, Figure 3.7 shows how the resync mechanism may re-synchronize the UART of a receiver. Even if it is not possible to correct the data byte, the mechanism will synchronize the receiver to the start of the following byte.

[3] Due to the encoding scheme used, a resync byte containing a 1-bit error is still distinguishable from other bytes.

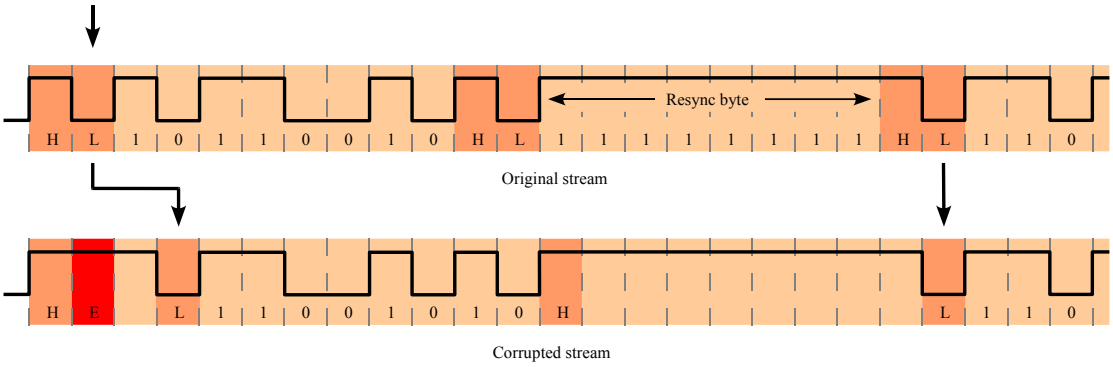


Figure 3.7: An original data stream and a corresponding corrupted stream using resync

The impact of resync on the entire transmission of a packet is shown in Figure 3.8. Resync bytes are used every n-th encoded byte ($n = 4$) and indicated with a ':'. Except once, all resync bytes have been detected correctly. Compared to Figure 3.5, the total amount of byte errors is much lower since the receiver could sooner or later be re-synchronized to the original data stream. For example, consider the fifth row of Figure 3.8. The actual resync byte (in this case the corrupted byte `0x5F`) is interpreted as a data byte since (i) it contains more than one single-bit error and (ii) the receiver is one byte behind the data stream. As the next original byte (`0xA5`, see Figure 3.5) is skipped due to Equation 3.1, the three following bytes are shifted one byte to the left (`0xA9`, `0x99`, `0x56`). After byte `0x56`, the next resync byte is recognized (':'). The receiver thus increases its byte position by one (indicated with `0x00`) and re-synchronizes itself to the original data stream.

```
01 7F AA 55 AA 55:59 A6 55 AA:96 A6 A9 5A:55 AA 55 A9:AA 65
99 A6:6A 56 A9 59:95 A9 65 96:AA AA 59 6A:5A 5A 96 99:5A AA
A6 5A:56 69 36 6A:56 6A 96 95:AA A5 56 59:55 9A 99 A5:A6 66
A5 A9:9A A6 59 96:66 6A AA 69:A5 99 AA 66:69 59 56 6A:56 96
95 69:69 95 95 A9:AA 56 59 5F A9 99 56:00 96 95 95 5A:99 55
A9 96:69 55 9A 66:56 99 9A 69:95 AA 69 59:66 9A 5A 6A:A9 AA
59 66:5A 99 AA A9:66 56 55 5A:56 AA 59 56:69 65 9A A5:56 95
A5 69:95 59 99 66:66 66 65 A5:AA A5 65 59:A9 56 56 9A:59 95
99 69:59 66 5A 55:96 69 6A 65:95 59 A9 59:9A 9A 56 A5:A5 56
96 66:A5 56 6A A9:6A A5 56 56:A9 95 A9 AA:66 59 9A 9A:55 AA
A6 69:96 65 9A AA:A9 A6 96 66:99 65 A1 8A:99 59 9A 99:56 A9
56 96:64 9A 99 65:65 AA 95 95:66 66 A5 59:99 A6 A5 A9:69 69
9A A6:61 56 5A 66:66 5A A9 69:95 59 65 65:65 9A A6 45:69 69
59 4B:96 6A:59 99:AA A6 55 99:A5 99 AA 6A:A5 A5 66 96:6A 6A
95 A6:96 69 56 59:A6 65 99 65:AA 4A 59 9A:69 A5 95 A5:69 69
A6 65:65 AA 99 66:65 5A 69 56:9A 99 A5 66:99 6A 96 69:95 6A
```

Figure 3.8: Part of an erroneously received packet using resync ($n = 4$)

3.2.3 Experimental Evaluation

To evaluate of our resync approach, we performed a comprehensive set of measurements. We placed 16 ESB nodes (including one *source* node) in line on our office floor, each at a distance of 1 m. A

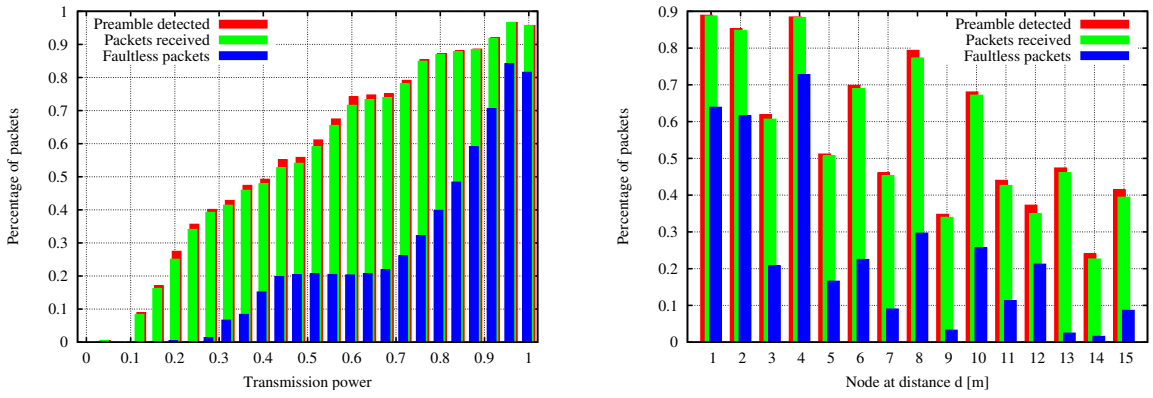
power supply unit was used for the source node, while all other nodes used rechargeable batteries. In doing so, we hope to get similar signal strengths for outgoing packets during each evaluation run.

In order to evaluate different transmission powers, the transmission power was increased from zero up to one in increments of 0.04. For each transmission power, the source node broadcast 100 data packets containing 256 bytes. Packet forwarding or routing was not employed. The data rate was set to 2 packets/sec. Nodes receiving a data packet did not send an acknowledgment but quietly updated their statistics. The content of each packet was predefined and known to all nodes a priori. In this way, the receiving nodes could simply keep book on the number of erroneously received packets as well as on the number of byte errors per packet.

At the end of each evaluation run, the source node collected all statistics stored at the receiving nodes. A notebook connected to the source node served as a database in order to support a convenient evaluation. We developed several *code modules* that operated as firmware plug-ins, e. g., to generate data packets, record packet statistics, perform statistical calculations and as a plug-in to collect the stored statistics from all nodes. Altogether, we have performed evaluation runs for different resync frequencies n, which was increased from zero (which means no resync was used at all) to a maximum value of 32.

Evaluation Results

Before we investigate the evaluation results of our resync approach, Figure 3.9(a) shows the average number of packets detected and received, as well as the number of correctly received packets for different transmission powers. At this time, the resync mechanism was not yet applied. As expected, the number of packets increases with a growing transmission power. However, if packets are sent with a transmission power of less than 0.28, the average number of correctly received packets is close to zero. While the difference between packets detected and received is very small, the difference between received packets and faultless packets is quite significant. Although the reception ratio of faultless packets increases with an increasing transmission power, the number of erroneous packets remains high. This is mainly due to the placement of nodes at different distances.



(a) For different transmission powers

(b) For different distances

Figure 3.9: Average packet reception without resync ($n = 0$)

The influence of a node's distance from the source node on the packets received is depicted in Figure 3.9(b), averaged over all transmission powers. Although the number of packets tends to decrease with increasing distance, there exists a high variation between adjacent nodes. This might be due to several reasons like (i) signal interferences, dispersions, or multi-path fading due to our indoor placement or (ii) transceiver calibration errors that are due to the low-cost, low-energy hardware. As other authors have reported in [276, 278, 282], we believe that the latter is the most likely explanation since we encountered such variations in the quality of nodes very often during preceding tests.

In the following, we now consider the improvements the resync mechanism is able to achieve. Figure 3.10 depicts the relative frequency with which an error occurred at a specific byte position within a packet. Due to the fact that some bits get lost if the UART misses a start bit (as was discussed in Section 3.2.1), it is likely that bytes near the end of a packet will be disturbed. This assumption is proven by Figure 3.10, where the error frequency increases heavily with the number of transmitted bytes per packet if no resync is used ($n = 0$).

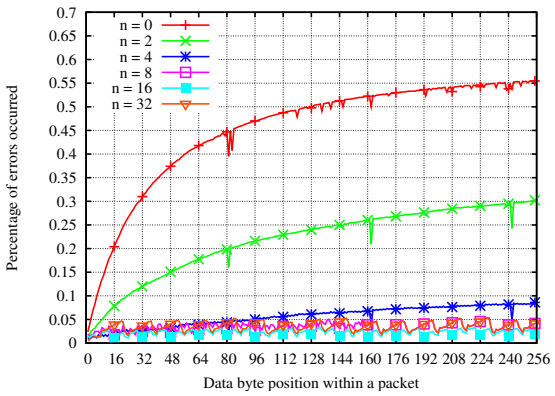


Figure 3.10: Number of errors occurring at a specific byte position

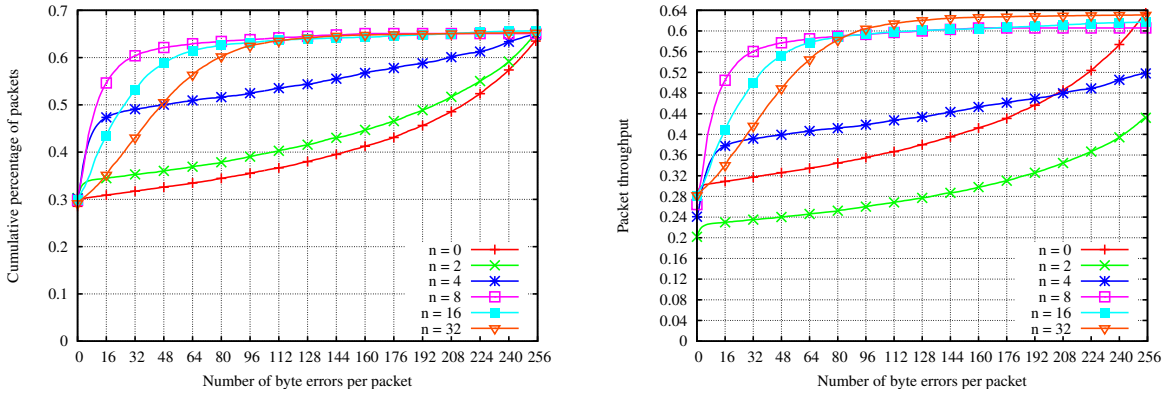
Interestingly, there are some peaks where the error frequency breaks down. For example, at byte positions 81 and 83, the error frequency is significantly lower than at adjacent positions. But how can the probability of an erroneous byte decrease in a stream of bytes? If caused by an implicit resync, the following bytes would have experienced fewer errors, too.

We performed a similar evaluation run with 50 zero bytes added to the beginning of the data stream. As a result, the peaks in Figure 3.10 move to the right by exactly these 50 bytes. Thus, the peaks must be due to the packet content itself.

A deeper analysis shows the following circumstance: Beginning with byte number 81, the data stream contains the following (non-encoded) bytes: `0xF4`, `0xC6`, `0xF4`, `0x2B`, `0x2F`, `0x56` etc. As we see, `0xF4` occurs twice in this stream. So if the stream shifts to the left by two bytes, byte number 81 would be coincidentally correct. For the second `0xF4` byte, we also find a similar bit pattern, the end of byte `0x2F` and the start of `0x56` (except for the last bit). Even if the receiver is out of sync (the start and stop bits are considered as data bits), the pattern remains the same. Thus, depending on the packet content, the peaks may move or even vanish completely.

As Figure 3.10 also illustrates, substantially better results are achievable using the resync mechanism as proposed in the previous section. However, particularly in terms of efficiency, it is essential to analyze how often resync bytes should be used to be optimal. The best results are achieved with $n = 16$, followed by $n = 32$, $n = 8$, $n = 4$, and $n = 2$. Thus, we can see that resync improves the error characteristics in any case. But the improvements are not monotonic with a decreasing frequency n. In fact, the results with $n = 64$ are worse than those with $n = 32$. That is due to the following trade-off: For $n \to \infty$, resync bytes are used sparsely, so they might have no effect. On the other hand, for $n \to 1$, resync bytes are used too often, leading to a byte stream that is no longer DC-balanced. As discussed in Section 3.2.1, the threshold used by the UART to distinguish between a high and a low signal is adapted to the ongoing transmission. That is, if too many bytes are transmitted due to too many resync bytes, the threshold will increase, and succeeding signals might be misinterpreted as zeros. Consequently, resync bytes might no longer be detectable and thus be ignored.

Figure 3.11(a) depicts the cumulative distribution of byte errors per received packet for different resync frequencies n. As a first observation, we see that using the resync mechanism improves the reception in all cases, independent of the resync frequency used.



(a) Cumulative distribution of byte errors per packet

(b) Packet throughout

Figure 3.11: Cumulative distribution of byte errors per packet and packet throughput

Concerning the number of errors per packet, most of the errors occur at the end of a packet for $n = 0$, as we have already seen from Figure 3.10. Thus, almost the entire packet is corrupted. The same applies to $n = 2$. However, for $n > 2$, we can observe how the error characteristics change. While for $n = 0$ about 31% of all received packets have fewer than 16 errors, the percentage increases to 47% for $n = 4$. For $n = 8$, it is even better; 55% of all received packets have fewer than 16 errors. However, for $n > 8$, the benefit of the resync decreases, as (i) a resync occurs less often, and (ii) the average number of erroneous bytes will increase until the receiver is synchronous again. In addition, the resync mechanism might skip some bytes due to Equation 3.1 if the UART at the receiver side has fallen behind[4]. Thus, for $n = 16$ and $n = 32$, more errors occur, even though the resync is successful.

So far we did not take into account the communication efficiency or the packet throughput defined as the ratio between bandwidth used and overhead cost. The bandwidth used relates to the number of transmitted bytes, including data as well as resync bytes. The costs are influenced by the frequency

[4] In the worst case, there are $\lceil \frac{n-1}{2} \rceil$ skipped bytes.

with which a resync occurs. For example, for $n = 2$, the efficiency is $1/2$ if one packet is considered[5]. According to Figure 3.11(a), Figure 3.11(b) shows the cumulative throughput for an increasing number of byte errors.

A resync frequency of $n = 2$ performs worst with respect to packet throughput since the number of resync bytes is quite high. It even performs worse than the case of using no resync at all. If fewer than 85 byte errors occur per packet, $n = 8$ achieves the best results. It is then outperformed by $n = 16$ and $n = 32$ since the overhead of both schemes is less, which leads to a better efficiency.

The main results of the evaluation are summarized in Table 3.1. In addition to the percentage of received packets and the appropriate throughput if up to e errors per packet are considered, the table shows the average number as well as the maximum number of byte errors, averaged over all packets and nodes. Based on these results, we can conclude that a resync frequency of $n = 8$ trades off the bandwidth usage and required resync cost better than other frequencies. Up to a reasonable number of errors, which might still be correctable by means of appropriate FEC codes [155, 192], it achieves the best performance. It is thus recommended if forward error correction is intended to be used in conjunction with the ESB platform. Otherwise, we do not need to make the effort of a period resync because, as it should be noted, the resync mechanism only reduces the number of bit errors per packet but is not able to avoid them completely.

n	Percentage of packets received and (throughput) per node with up to e errors						Errors per packet and node	
	$e = 0$		$e \leq 16$		$e \leq 32$		Average	Maximum
0	0.29	(0.29)	0.31	(0.31)	0.32	(0.32)	31	96
2	0.30	(0.20)	0.34	(0.23)	0.35	(0.24)	28	83
4	0.30	(0.24)	0.47	(0.38)	0.49	(0.39)	20	64
8	0.30	(0.26)	0.55	(0.51)	0.60	(0.56)	12	32
16	0.30	(0.28)	0.43	(0.41)	0.52	(0.50)	24	70
32	0.29	(0.28)	0.35	(0.34)	0.43	(0.42)	28	79

Table 3.1: Error characteristics for different resync frequencies

3.2.4 Conclusions

In this first part, we have analyzed the wireless communication characteristics of the ESB platform in more detail. Due to the hardware design, specific communication errors might occur if the sender and the receiver get out of sync during a transmission. In this case, the following bits will be misinterpreted at the receiver side. We hence proposed a periodic resync mechanism that is able to re-synchronize the sender's and receiver's state machines and thus reduce the number of errors per packet significantly.

We have reported our results to the researchers at the FU Berlin and pointed out the design problems of using this radio transceiver in conjunction with UARTs. Although our resync mechanism reduces the impact of the UART on the reception performance, a more preferable solution would be to change the hardware design, rather than trying to fix the synchronization problem in software. While UARTs are well-suited for *wired* communications with loss rates that are orders of magnitude lower, they are

[5] In general, the efficiency is $1/n$ for a single packet.

unsuited for *wireless* devices. Meanwhile, many manufacturers rely on *synchronized* communication components, such as the recently launched next generation of the ESB [210]. Due to a precise timing, UARTs no longer need be employed, avoiding the framing of bytes by start and stop bits.

The next section investigates how FEC performs in conjunction with the proposed resync mechanism and to what degree FEC is able to improve the packet delivery ratio of ESB nodes in practice.

3.3 Forward Error Correction

In traditional 802.x-based Wireless LANs, the strength of a radio signal will usually be sufficient to reliably cover a relevant area [95]. If not, a larger number of base stations can easily be used. Thus, many evaluations come to the conclusion that incorrect packets occur rarely; if they occur at all, they are caused by burst errors due to disturbing signal sources [141]. Here, error recovery by retransmission [242] is more efficient than adding redundant information to every packet in advance.

However, Zhou *et. al* [278] have shown that radio communication over large distances or with weak radio signals (as it is expected in sensor networks) exhibit different error characteristics. Single bit errors occur with a much lower variance, which means that they are more evenly distributed. For example, consider a packet with a size of 256 bytes. If we assume that there is a 50% chance for a single bit error within 500 bits, the likelihood of receiving an error-free packet of 256 bytes is as low as 6%. Despite the small number of single bit errors, we would have to retransmit more than 16 packets on average in order to get one through. On the other hand, the small number of bit errors could be compensated easily by FEC, which motivates the usage of *proactive* techniques rather than using ARQ with *reactive* retransmissions.

Of course, proactive and reactive techniques can be combined into a hybrid approach of FEC and ARQ as in [243], where the first packet contains no redundant information, and retransmitted packets include information for forward error correction. Since the FEC code size depends heavily on the error characteristics of the underlying wireless channel, adaptive FEC code schemes are also proposed [9]. Adaptive codes determine the appropriate code size dynamically based on the number of arrived acknowledgements, the bit error rate, or the signal-to-noise ratio.

Another hybrid-ARQ technique is described by Zhao and Valenti in [275], where retransmitted packets do not need to come from the same source node. Instead, relay nodes that overheard the transmission may transmit lost packets. The idea of hybrid ARQ is that corrupted packets are buffered at the receiver and combined with retransmitted packets. However, rather than sending the original packet again, the sender may transmit an encoded version that can then be combined with the corrupted packet sent before. Dubois-Ferrière *et al.* [78] present such a packet combining scheme by using plain and parity packets and evaluate the performance in a multi-hop sensor network. Exploiting the reception of corrupted packets significantly improves the network performance and reduces the number of packet transmissions along a multi-hop route. Especially for sensor networks, combining plain and parity packets provides a simple form of adaptive coding which does not rely on any channel measurements.

Concerning the ESB platform, Willig and Mitschke [257] performed several measurements and analyzed the bit error rate over a long period of time. However, due to the synchronization problem we described in the first part of this chapter, they encountered burst errors in the majority of cases. Since most of these burst errors cannot be corrected by common FEC codes, they rather recommended the use of ARQ. However, they were not aware of the synchronization problem caused by the UARTs. Thus, this section discusses the usage of FEC if resync is applied.

3.3.1 Forward Error Correction Codes

Before we evaluate different FEC codes on the ESB platform in conjunction with resync, we first present some theoretical aspects of FEC. We consider three types of codes that are able to correct different kinds of errors: A code that can correct single bit errors, one that is able to correct double bit errors, and a Reed-Solomon code that recovers so-called *symbol* errors, depending on the redundancy of the code. We also describe the concept of *interleaving*, which can be used to spread burst errors within a packet. A comprehensive overview of FEC codes, also in comparison with ARQ techniques, can be found in [130, 155, 156, 192].

Linear Block Codes

Let $\mathbb{B} = \{0, 1\}$ and $n \in \mathbb{N}$. Then a code $C \subseteq \mathbb{B}^n$ will be called a *linear block code* if the sum of two codewords u and v is a codeword, too. Let $k \in \mathbb{N}$ be the dimension of C with $\{b_1, \ldots, b_k\}$ being a basis of C. The $k \times n$ matrix

$$G = [b_1 \ldots b_k]^T \tag{3.2}$$

is called the *generator matrix* of the code C since C is constructed by G with

$$C = \{xG \mid x \in \mathbb{B}^k\}. \tag{3.3}$$

Thus, C is called an (n, k) code, where k refers to the number of data bits and n refers to the number of bits the encoded codeword has. With I_k being the $k \times k$ identity matrix, G can be transformed to $[I_k P]$, where P is a $k \times r$ binary matrix and r refers to the number of *parity bits* that are added to the code C. Using the generator matrix G, the *parity matrix* H is constructed and is of the form $[P^T I_r]$. Then, for all codewords $x \in \mathbb{B}^n$, we get

$$x \in C \quad \Leftrightarrow \quad xH^T = 0. \tag{3.4}$$

xH^T is called the *syndrome* s of x. If $s \neq 0$, an error occurred during the encoding process.

Hamming Codes

Hamming codes are linear (n, k) codes that are able to correct 1-bit errors [107]. The parity matrix H can be constructed in a simple way by setting the j-th column vector to the binary form of number j.

For example, for the $(7, 4)$ Hamming code, H would be

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}, \tag{3.5}$$

with the generator matrix G being

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.6}$$

Note that H and G are not of the form $[P^T I_r]$, respectively $[I_k P]$, but could easily be transformed into it by exchanging some column vectors.

If a 1-bit error occurs, the syndrome s gives the position of the erroneous bit. For example, let $c \in \mathbb{B}_n$ be a valid codeword that is received as $x = c + e$ with a 1-bit error vector $e \in \mathbb{B}_n$. Then, we get

$$s = xH^T = (c + e)H^T = eH^T. \tag{3.7}$$

Since eH^T corresponds to the e-th column in H, s gives the position of the error.

As the Hamming code is able to correct 1-bit errors and detect up to 2-bit errors, it is referred to as a *single error correction* and *double error detecting* (SEC-DED) code. Other SEC-DED codes are for example the one presented in [215] or the *odd-weight-column* codes [117], where each column of H has an odd *weight*, i. e., the number of ones is odd. By using minimum odd-weight-column codes, the number of ones contained in the H matrix is further minimized. In corresponding circuitry, this requires less hardware area. Ghosh *et al.* [96] have shown that the power consumption can be reduced by exploiting the degree of freedom in selecting the parity matrix. Thus, odd-weight-column codes are most relevant when error correction is implemented in hardware.

Double Error Correction Codes

A *double error correction* and *triple error detection* (DEC-TED) $(16, 8)$ code was proposed by Gulliver and Bhargava [102]. With P defined as

$$P = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}, \tag{3.8}$$

the generator matrix G is given in its standard form $[I_8 P]$ and the parity matrix H as $[P^T I_8]$. If a 1-bit or 2-bit error occurs, the syndrome s is either equal to a single column of H or to an additive (exclusive-or) combination of two columns. The index of the involved columns then represents the position of the bit error(s).

Interleaving

In order to combat the effects of burst errors, *interleaving* can be used. In this way, two or more codewords are interleaved before they are transmitted. The number of interleaved codewords k refers to the *depth* of an interleaver. The interleaver first stores the k codewords of size m in a $k \times m$ buffer row-by-row. We thus call such an interleaver a (k, m)-interleaver. It then outputs the interleaved codewords column by column. Thus, in the output stream there are always $k - 1$ other bits between two successive codeword bits. At the other end, a deinterleaver works in the reverse way.

If the interleaving depth is sufficiently large, the correlation between two successive codeword bits will be minimized. The deinterleaver might thus have enough capability to decode the codeword successfully. As shown in [280], interleaving is effective if tk exceeds the average burst length, where t denotes the number of correctable errors. For example, consider a stream of 16 bytes where the first two bytes are completely disturbed, after the interleaving was performed. Using an $(8, 8)$-deinterleaver would reconstruct an output stream with 16 bytes where 8 bytes contain a 2-bit error. On the other hand, a $(16, 8)$-deinterleaver would produce an output stream with 16 bytes where all bytes contain a 1-bit error only.

Figure 3.12 illustrates this example. Bit errors are indicated in red. The first eight output bytes are represented by solid arrows, while the second eight bytes are shown by dashed arrows. Even if both times the first and the second bytes in the output stream are completely destroyed, only the $(16, 8)$-deinterleaver is able to spread the bit errors over all 16 bytes. The $(8, 8)$-deinterleaver assigns the first two bytes to the same decoded byte block due to its lower interleaving depth and thus produces eight bytes containing 2-bit errors and eight bytes containing no bit error.

Reed-Solomon Codes

Today, Reed-Solomon (RS) codes [199] are widely used, e. g., in digital television, wireless and satellite communication, broadband modems, CD's and DVD's, etc. Like the before-mentioned FEC codes, they work by adding some redundancy to the original data, which is later used to correct a certain number of errors. However, RS codes do not correct multiple single bit errors but complete *symbols*, which contain a fixed number of bits. As before, the number of correctable errors mainly depends on the amount of information added.

RS codes belong to the class of *systematic linear block* codes. Systematic means that the encoded data consist of the original data and some redundant symbols added to the end (called the *code block*). Each block is divided into multiple m-bit symbols with a symbol size of typically 3 to 8 bits. Finally, the linearity property of the code ensures that every possible m-bit word is valid for encoding.

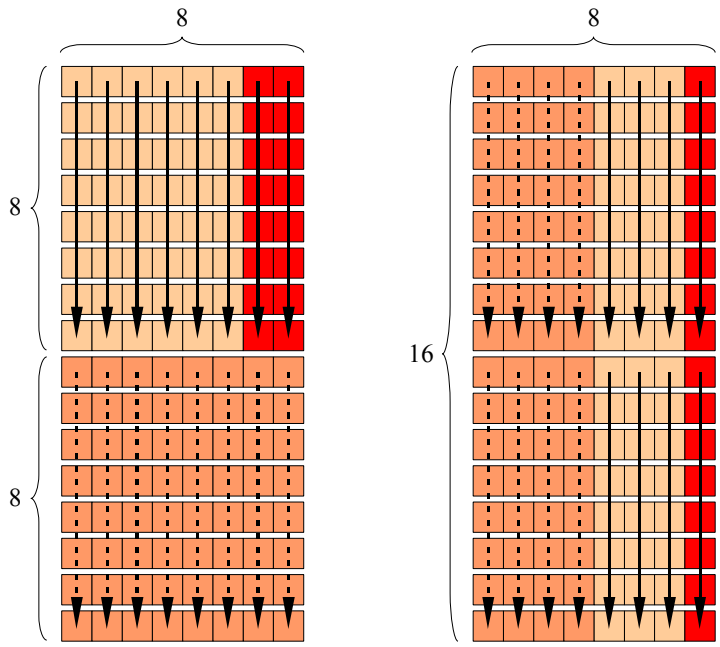


Figure 3.12: Example of two $(8, 8)$-interleavers and one $(16, 8)$-interleaver

An RS(n, k) code then specifies a code with n encoded m-bit symbols per code block. The number of original symbols is specified by k, thus $n - k$ refers to the amount of redundancy used. Given the symbol size m, the maximum length of the RS code, which is denoted by n, is $2^m - 1$.

With $2t = n - k$, an RS decoder is able to correct up to t symbol errors. That means that either only t bits (single bit errors) or up to tm bits (all symbol bits are erroneous) may be corrupted, depending on how many symbols are affected. Therefore, RS codes are able to correct burst errors of up to t unknown symbols. If the position of a symbol error is known, such an error is called an *erasure*. Erasures are easier to correct since their positions need not be identified. Thus, a codeword containing r unknown symbol errors and s erasures can still be recovered if $2r + s \leq 2t$.

A widely used RS code is the RS$(255, 223)$ code with 8-bit symbols. Each codeword has a length of 255 bytes, of which 223 bytes are data and 32 bytes are parity bytes. If not all data bytes are required, the code may be shortened by regarding missing data bytes as zero symbols. Zero symbols need not be transmitted and are re-inserted at the receiving side prior to decoding. For example, an RS$(255, 223)$ code can be shortened to an RS$(193, 161)$ code by adding 62 zero bytes to the data. Then, the data are encoded into an RS$(255, 223)$ codeword, of which only 193 original data bytes and the 32 parity bytes are transmitted.

The encoding and decoding processes of RS codes are based on a mathematical construct known as *finite fields* or *Galois fields*. A mathematical field is finite if the result of an arithmetic operation (like $+, -, \cdot, /$) on a field element is an element of the field itself. The encoder of an RS code then generates a codeword based on these operations by using a special *generator polynomial*. Each codeword has the property that it is exactly divisible by the polynomial. The $2t$ parity symbols are finally given by the last $2t$ bits of the codeword.

At the receiving side, the codeword can be considered to be a polynomial, too. The decoder carries out several steps in order to recover the possibly corrupted data. At first, the $2t$ syndromes of the codeword are calculated. The roots of the generator polynomial are substituted into the polynomial of the received codeword. Then, the location of the symbol errors need to be found, which requires solving simultaneous equations with t unknowns. An error locator polynomial is then created by using the Berlekamp-Massey algorithm [22, 169]. The roots of the locator polynomial indicate the locations of symbol errors, which can be calculated by the Chien search algorithm [57]. After the locations are known, symbol errors can be corrected by the Forney algorithm [85]. This again involves solving simultaneous equations with t unknowns. Further information concerning encoding and decoding of RS codes can be found in [155] or [192].

3.3.2 Analyses of FEC Codes

We can now analyze the performance of the presented FEC codes concerning the *packet delivery ratio*, the *packet delivery cost*, and the *utilization* defined as the fraction of carried information per delivered bit. Let p be the bit error rate on the wireless channel, k be the number of data bytes, and n be the number of bytes the encoded packet finally contains. Furthermore, let X be a random variable denoting the number of bit errors per codeword. X follows the binomial distribution with parameters $\hat{n}$ and p, i. e., $X \sim B(\hat{n}, p)$. The probability mass function for getting exactly $\hat{k}$ successes is then given as

$$f(\hat{k}, \hat{n}, p) = \binom{\hat{n}}{\hat{k}} p^{\hat{k}} (1-p)^{\hat{n}-\hat{k}}, \tag{3.9}$$

and the cumulative distribution function is expressed as

$$F(x, \hat{n}, p) = \sum_{j=0}^{\lfloor x \rfloor} f(j, \hat{n}, p). \tag{3.10}$$

Packet Delivery Ratio

The probability r of delivering a packet successfully if no FEC coding is used is

$$r(p, k) = (1-p)^{8k}. \tag{3.11}$$

If in addition ARQ with R retransmissions is employed, the probability changes to

$$r(p, k, R) = 1 - (1 - r(p, k))^{R+1}. \tag{3.12}$$

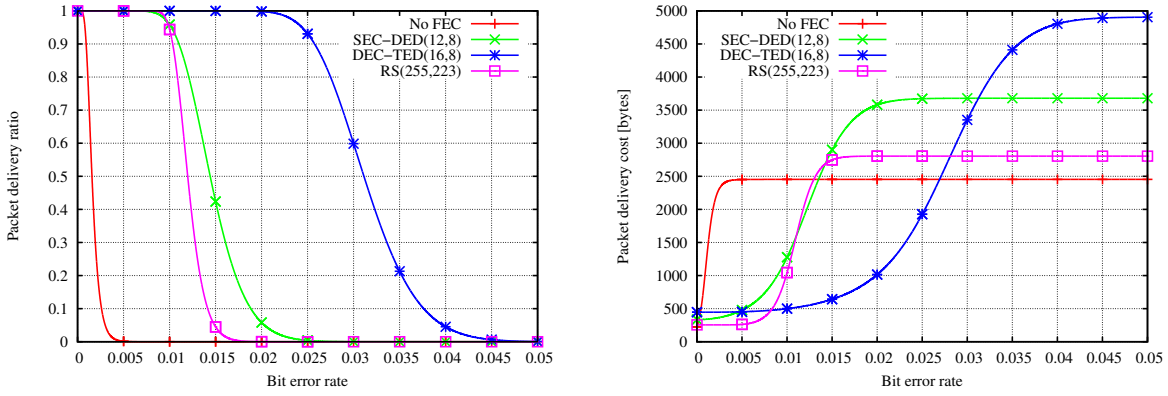
Concerning a SEC-DED$(12, 8)$, a DEC-TED$(16, 8)$, and an RS$(225, 233)$ code, the packet delivery ratio can be calculated by

$$r_{SEC}(p, k, R) \quad = \quad 1 - (1 - F(1, 12, p)^k)^{R+1}, \tag{3.13}$$

$$r_{DEC}(p,k,R) = 1-(1-F(2,16,p)^k)^{R+1}, \tag{3.14}$$

$$r_{RS}(p,k,R) = 1-(1-F(16,k+32,1-(1-p)^8))^{R+1}. \tag{3.15}$$

Figure 3.13(a) shows the packet delivery ratio for different bit error rates, using up to 10 retransmissions. If no FEC coding is used, the delivery ratio heavily decreases with an increasing bit error rate. By using an RS$(255, 223)$ code, the delivery ratio improves significantly. However, it tends to decrease for a bit error rate above 10^{-3}, which already causes more than 20 bit errors per packet. Further improvements are possible by using an SEC-DED$(12, 8)$ and DEC-TED$(16, 8)$ code, where the DEC-TED code outperforms all other schemes. Up to a bit error rate of $2 \cdot 10^{-3}$, which is equal to 71 bit errors per packet, the packet delivery ratio remains nearly 100%. In contrast, the delivery ratio of the RS$(255, 223)$ code is already equal to zero. However, note that bit errors were distributed uniformly. Thus, as soon as burst errors occur, the performance of the DEC-TED$(16, 8)$ code might be worse.



(a) Packet delivery ratio

(b) Packet delivery cost

Figure 3.13: Packet delivery ratio and cost for different FEC codes ($k = 223$, $R = 10$)

Packet Delivery Cost

Although the DEC-TED$(16, 8)$ code shows a significantly better packet delivery ratio, the cost of achieving this result should not be neglected. Here, the cost represents the number of bytes that must be transmitted over the wireless channel in order to deliver a packet successfully. For example, a packet containing k data bytes causes a cost of k bytes if no FEC coding is used, but $2k$ if the DEC-TED$(16, 8)$ code is employed. Moreover, if erroneous packets are retransmitted, the cost might be even higher.

With q being the probability of an erroneous packet, we can calculate the expected number of transmissions t as follows:

$$t(q,R) = q + 2q(1-q) + \ldots + (R+1)q^R(1-q) + (R+1)q^{R+1}$$

$$
\begin{aligned}
&= (1-q)\sum_{i=0}^{R}(i+1)q^i + (R+1)q^{R+1} \qquad (3.16)\\
&= \frac{q\left(1-(R+1)q^R+Rq^{R+1}\right)}{1-q} + 1 - q^{R+1} + (R+1)q^{R+1}.
\end{aligned}
$$

The costs for non-coding and the three FEC codes are then given as

$$
\begin{aligned}
c(p,k,R) &= t(r(p,k,R),R)\cdot k, && (3.17)\\
c_{SEC}(p,k,R) &= t(r_{SEC}(p,k,R),R)\cdot 3/2k, && (3.18)\\
c_{DEC}(p,k,R) &= t(r_{DEC}(p,k,R),R)\cdot 2k, && (3.19)\\
c_{RS}(p,k,R) &= t(r_{RS}(p,k,R),R)\cdot (k+32). && (3.20)
\end{aligned}
$$

Figure 3.13(b) illustrates the packet delivery cost if up to 10 retransmissions are used. If the bit error probability is zero, the DEC-TED$(16,8)$ code causes the highest cost since for each data byte one (in this case unnecessary) parity byte is generated. For this case, using no FEC at all performs best, as redundant bytes are not required but only cause overhead. However, as soon as the bit error rate increases, the delivery cost climbs due to employed retransmissions. Since the number of retransmissions is limited to 10, at most $11 \cdot 223$ bytes are transmitted. The lowest costs are achieved by the RS$(255,223)$ code for a bit error rate below $8 \cdot 10^{-4}$. Beyond that value, the costs increase since the likelihood of packets being retransmitted grows. From this point on, the DEC-TED$(16,8)$ code shows the best performance, as retransmission of the complete packet can often be avoided. However, if the bit error rate is greater than about $2.7 \cdot 10^{-3}$, FEC becomes more expensive than non-coding. The SEC-DED$(12,8)$ always shows the worst performance, as it is either outperformed by the RS$(255,223)$ or the DEC-TED$(16,8)$ code.

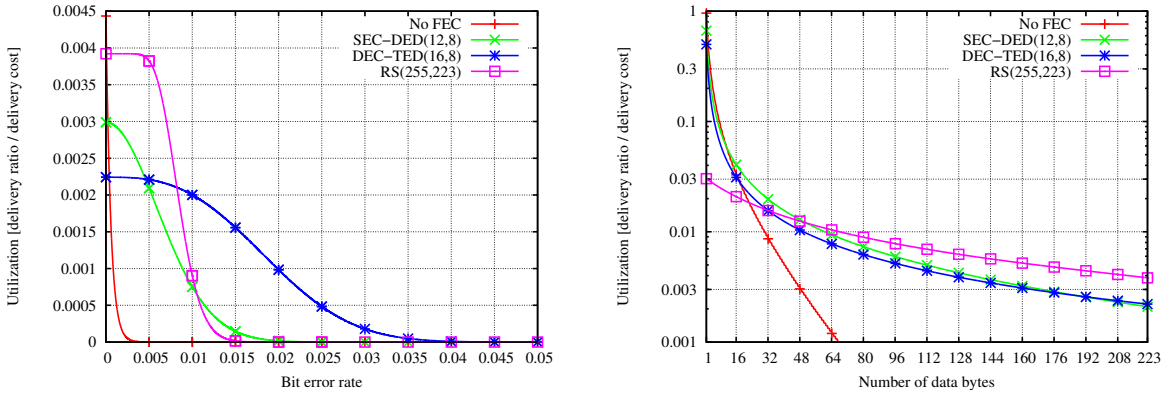
Utilization

As it has been shown, there is a trade-off between the packet delivery ratio and its corresponding cost. This trade-off is taken into account by the utilization, which is defined by the fraction of delivery ratio and delivery cost. Thus, they can be calculated as

$$
\begin{aligned}
u(p,k,R) &= r(p,k,R)/c(p,k,R), && (3.21)\\
u_{SEC}(p,k,R) &= r_{SEC}(p,k,R)/c_{SEC}(p,k,R), && (3.22)\\
u_{DEC}(p,k,R) &= r_{DEC}(p,k,R)/c_{DEC}(p,k,R), && (3.23)\\
u_{RS}(p,k,R) &= r_{RS}(p,k,R)/c_{RS}(p,k,R). && (3.24)
\end{aligned}
$$

According to Figure 3.13, Figure 3.14(a) shows the utilization function of the different FEC codes. As long as the bit error rate is smaller than 10^{-5}, employing FEC coding is useless, since its slightly better delivery ratio does not compensate for its redundancy. However, if the bit error rate is greater than 10^{-5}, FEC is able to achieve a better utilization. While the RS$(255,223)$ code performs best for a bit error rate of up to $8 \cdot 10^{-4}$, it is outperformed by the DEC-TED$(16,8)$ code due to higher delivery

ratios. As Figure 3.13(b) has shown, the SEC-DED$(12, 8)$ code has no practical relevance for large k since either the RS$(255, 223)$ or the DEC-TED$(16, 8)$ code performs better.



(a) Different bit error rates

(b) Different data sizes

Figure 3.14: Utilization of different FEC codes ($p = 5 \cdot 10^{-4}$, $R = 10$)

For a bit error rate of $5 \cdot 10^{-4}$, the impact of the number of data bytes k on the utilization is illustrated in Figure 3.14(b)[6]. Due to a fixed redundancy of 32 bytes, the RS$(255, 223)$ code shows the lowest utilization if only few data bytes are encoded. In this case, using no FEC performs best. For a data size between 10 and 51 bytes, the SEC-DED$(12, 8)$ performs better for the first time. However, above those sizes, the RS$(255, 223)$ code compensates for its fixed amount of redundancy and outperforms all other schemes.

3.3.3 Experimental Evaluation

Motivated by the theoretical results obtained in the previous sections, we now evaluate the presented FEC codes by means of real experiments. The evaluation setup is similar to the one used in Section 3.2.3. Again we placed 16 ESB nodes (including the source node) in line on our office floor, each at a distance of one meter, and increased the transmission power from zero to one in increments of 0.04. Each time, the source node broadcast 100 data packets of 255 bytes each. In order to capture errors in the data, a random but fixed content was used during all evaluation runs. The data rate was set to 2 packets/sec.

Upon broadcasting a data packet, the source node polled each node to get information about packet reception ratios. The polling was performed with the maximum transmission power and repeated until all information was received correctly. Nodes that were polled by the source node answered with the data currently contained in the receiving buffer. Due to the node's limited memory space, the source node transmitted the polled information to a notebook over a serial connection, which logged all data in a database for later processing.

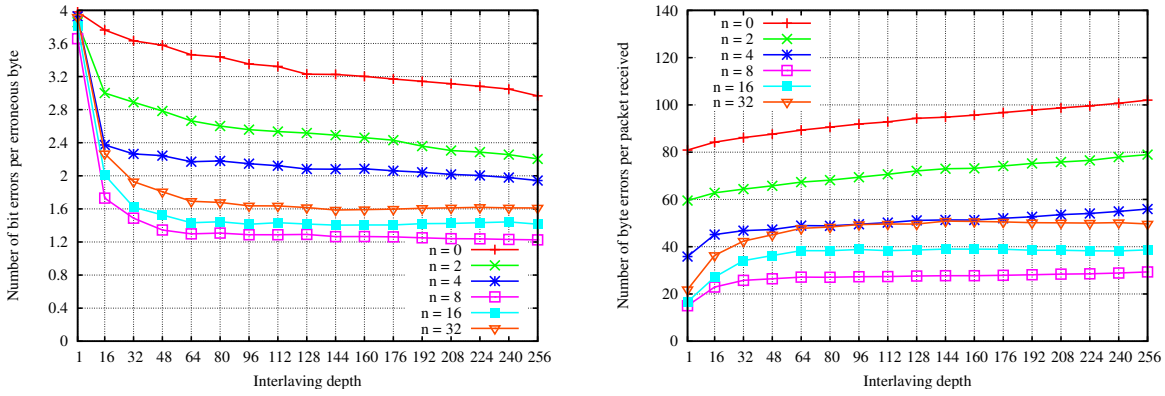
We have performed several evaluation runs with different resync frequencies, which likely influence the reception characteristics remarkably. In addition, we analyzed the impact of interleaving by using

[6]Note the logarithmic scale on the y-axis.

different interleaving depths as described in Section 3.3.1. For each generated log file, the reception performance of one specific FEC code was evaluated. By using the same log file for all FEC codes together, the results thus relied on the same error characteristics and offered better comparability.

Evaluation Results

At first, we will analyze the number of bit errors per erroneous byte, which is shown in Figure 3.15(a) for different resync frequencies n and an increasing interleaving depth k. We see that resync has only a small impact if no interleaving is used. The number of bit errors is close to the average of four bit errors per byte, which is equal to the mathematical expectation. If the interleaving depth increases, bit as well as burst errors occur and are spread over the packet more effectively. Although more bytes may get corrupted, the average number of bit errors per erroneous byte decreases. Because the number of byte errors per packet is minimal for $n = 8$ and increases for $n \to 1$ and $n \to \infty$ (see Section 3.2.3), Figure 3.15(a) shows a similar effect concerning the number of bit errors. For $n = 8$ and $k \geq 64$, the number of bit errors decreases to 1.2 bit errors on average. This is an improvement of about 60% compared to $n = 0$, where 2.8 bit errors occur per erroneous byte.



(a) Bit errors per erroneous byte

(b) Byte errors per packet received

Figure 3.15: Bit and byte errors for increasing interleaving depths

How much the number of byte errors increases if interleaving is used is shown in Figure 3.15(b). On average, a packet contains about 80 bytes with one or more bit errors if no resync and interleaving are employed. While for $n = 8$ and $k = 0$, the number of errors per packet can be reduced to about 18, $n \neq 8$ leads to a worse performance, as synchronizations are performed either too often or too rarely. However, increasing the interleaving depth k affects the number of byte errors per packet positively. Since interleaving is not able to reduce the total number of bit errors per packet but only spreads burst errors, the number of bytes containing at least one bit error grows for $k \to 256$.

At this point we could already assume that interleaving may be advantageous for both the SEC-DED$(12, 8)$ and the DEC-TED$(16, 8)$ code since the number of bit errors per erroneous byte is reduced significantly. However, we should note that Figure 3.15(a) only shows the number of bit errors per erroneous *byte* and not per erroneous *codeword*, which may consist of either 12 or 16 bits. In addition, in order to correct all errors per packet, an average of 1.2 bit errors per byte may already be

too high for the SEC-DED$(12, 8)$ code, which can only recover codewords containing no more than one bit error.

While interleaving might improve the recovery ratio of the SEC-DED$(12, 8)$ and DEC-TED$(16, 8)$ codes, it certainly affects the RS$(255, 223)$ code adversely because interleaving increases the total number of erroneous bytes per packet. However, the RS$(255, 223)$ code is only able to correct up to 16 unknown symbols, independent of how many bit errors occurred per symbol. Thus, in this case spreading burst errors by interleaving is neither useful nor necessary.

According to Figure 3.15(a) and 3.15(b), Figure 3.16 illustrates how the number of bit errors per erroneous byte are distributed for the special case of $n = 8$. Without interleaving, about 90% of erroneous bytes contain at least a 1-bit error, respectively 80% contain at least a 2-bit error that would be uncorrectable if SEC-DED$(12, 8)$ coding were used. For $k \rightarrow 256$, the percentage of erroneous bytes containing at least a 2-bit error is reduced to about 47%. However, as noted above, SEC-DED$(12, 8)$ coding allows only a 1-bit error among 12 bits, respectively a 2-bit error among 16 bits for DEC-TED$(16, 8)$ coding. Moreover, *each* codeword within a packet must be correctable in order to decode the complete packet successfully. If only one codeword contains more errors, the complete packet cannot be decoded at all.

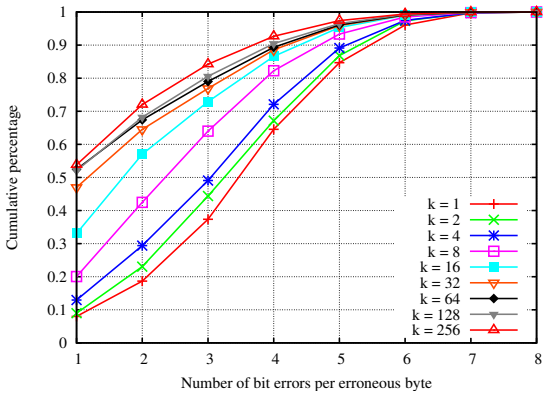
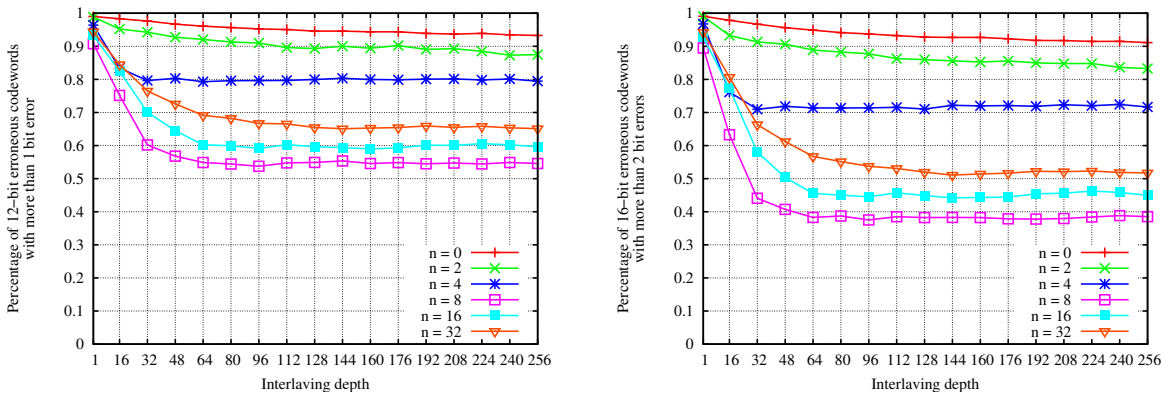


Figure 3.16: Cumulative distribution of bit errors per erroneous byte ($n = 8$)

The percentage of 12-bit codewords that contain more than one bit error is depicted in Figure 3.17(a); Figure 3.17(b) shows the percentage of uncorrectable DEC-TED$(16, 8)$ codewords. Again, the best results are achieved for a resync frequency of $n = 8$. If the interleaving depth k is greater than 64, about 55% of erroneous SEC-DED$(12, 8)$ codewords will be uncorrectable. Compared to Figure 3.16, that is an increase of about 8%. Concerning DEC-TED$(16, 8)$ coding, only 39% of erroneous codewords cannot be recovered, which is an improvement of about 16% compared to the SEC-DED$(12, 8)$ code.

Finally, Figure 3.18 depicts the packet delivery ratio of FEC coding using the best-evaluated resync frequency of $n = 8$. For comparison, also the packet delivery ratio of non-coding is shown. The best result is achieved by the RS$(255, 223)$ code. Note that the packet delivery ratio is averaged over all nodes and all evaluated transmission powers. Thus, the overall performance includes long-distance as well as low-power transmissions. The delivery ratio of RS$(255, 223)$ coding could be further improved



(a) SEC-DED$(12, 8)$ code (b) DEC-TED$(16, 8)$ code

Figure 3.17: Percentage of uncorrectable codewords

if resync information regarding erasure positions are used[7]. In this way, even more than 16 symbols could be corrected.

Since interleaving is not able to reduce the number of bit errors on its own, it has no effect on the packet delivery ratio if FEC is not employed. As interleaving increases the number of erroneous bytes, it worsens the performance of RS$(255, 223)$ coding. On the other hand, the DEC-TED$(16, 8)$ code benefits from interleaving significantly, resulting almost in the same packet delivery ratio as the RS$(255, 223)$ code. However, the impact of interleaving on the SEC-DED$(12, 8)$ code is marginal, although the percentage of correctable codewords increases. But even if a codeword contains fewer than two bit errors on average, it is possible that no packet will be recovered, because any 2-bit error contained in one codeword already prevents the entire packet from being corrected.

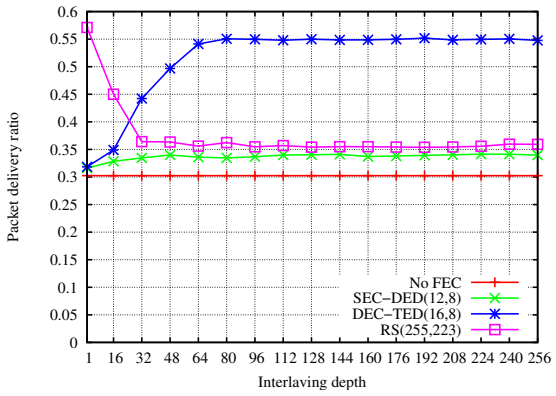


Figure 3.18: Packet delivery ratio of different FEC codes ($n = 8$)

While the delivery ratio of non-coding is improved in any case, note that the overheads of the considered FEC codes are quite different. While RS$(255, 223)$ coding requires 32 additional bytes, independent of the number of data bytes, the SEC-DED$(12, 8)$ and DEC-TED$(16, 8)$ codes need a redundancy

[7] In case a byte position within a packet is skipped due to the synchronization problem, the skipped byte can be considered as an erasure.

of 50% and 100%, respectively. That is, for a data size of 223 bytes as used in our evaluation setup, 112, respectively 223 bytes are redundant and cannot be used to carry any application data.

3.3.4 Conclusions

In conclusion, we have seen that significant improvements in the packet delivery ratio are possible if FEC codes are used in conjunction with a periodic resync mechanism. As long as bit errors are uniformly distributed, DEC-TED$(16, 8)$ coding outperforms RS$(255, 223)$ codes for high bit error rates. However, in the presence of burst errors and larger packets, RS$(255, 223)$ coding is more efficient and uses less redundancy. On the other hand, SEC-DED$(12, 8)$ coding actually cannot be recommended. Even with interleaving, the probability that a codeword will contain more than one bit error is too high to correct an erroneous packet completely.

In the experimental evaluation, the DEC-TED$(16, 8)$ code achieved nearly the same packet delivery ratio as RS$(255, 223)$ coding if bit interleaving was applied. While RS codes are quite complex and required much memory and processing time, DEC-TED$(16, 8)$ codes are easier to implement but needed twice as many data bytes for redundancy.

Thus, whether or not FEC should generally be used heavily depends on the environment a sensor network is deployed in, and on its application. For reliable data communication as well as in lossy environments it is surely suitable, due to a better channel utilization than with ARQ. However, the fixed amount of redundancy added to a transmitted packet is actually dissatisfying. While low error rates require less redundancy, more redundancy is needed if the error rate starts to increase. None of the FEC codes presented so far are able to adapt to such changes.

One solution to this problem is *fountain* codes, which we consider in the next section. They are also particularly interesting if bulk data need to be disseminated to multiple receivers, further improving the performance of non-coding as well as that of RS codes.

3.4 Data Dissemination Using FEC Coding

Commonly, a large amount of data is split into small-sized chunks before it is disseminated through a network. If the data is additionally encoded by an FEC code, missed chunks can easily be recovered at the receiving side. Fountain codes are a special kind of FEC code which have the property that the sender provides the data in a virtually endless stream by combining original chunks at random. No matter which chunks get lost, each receiver only needs *any* k chunks from the stream, with k being the number of chunks the data block consists of. Particularly in broadcast scenarios, fountains have the advantage that only little redundancy is required, even if several receivers missed different chunks.

For example, consider a network of wireless sensor nodes that are capable of sensing, processing, and communicating [11, 67]. In general, the detection of an *event* or a particular *stimulus* is first processed by a node and then forwarded to one or several sink nodes in the network. Besides this n-to-1 communication, also a 1-to-n communication is required where the sink sends queries or control

packets to a fraction of, or to all sensor nodes. Since it is likely that the nodes will be scattered over a wide area and therefore unable to communicate with each other directly, intermediate nodes forward messages in a multi-hop fashion.

In particular, we consider the case of bulk data transmission, as it is necessary for large queries that, e. g., carry image information for video sensor nodes [132, 137, 195], code updates [149], or new firmware releases [121, 138]. Sending small-sized chunks over the network has the advantage that in case of loss only lost or erroneous chunks must be resent. However, if the data is broadcast to multiple receivers, it is likely that most of the retransmitted chunks will be redundant to many nodes. Minimizing the number of redundant packets would reduce the energy costs of the sensor network significantly and thus extend its operational lifetime.

One possibility to achieve a reduction in the number of packet transmissions is to use specific coding schemes. At first, all original chunks are encoded by the sender before they are broadcast to one or more receivers. If chunks get lost during the transmission, additional chunks are sent such that all receivers are able to recover missed parts and finally decode the data. The advantage of encoding is that additional chunks may be useful to more than one receiver at the same time, even if the receivers have missed different chunks. Note that in this case forward error correction is not intended to correct bit errors in single chunks but to recover completely missing ones. As mentioned before, such losses are termed *erasures*.

A classic erasure correction code is the RS(n, k) code we have considered in the previous section. A receiver of such a block code would be able to decode the original k symbols if *any* k of the transmitted n symbols are received without errors[8]. Thus, even if two receivers might have missed different symbols, an additional one might already be sufficient for both to decode the original ones successfully. However, a disadvantage of Reed-Solomon codes is, besides their high computational complexity, their fixed redundancy $n - k$, which limits the number of symbols that may get lost.

Fountain codes [45, 158, 163] overcome this disadvantage by providing an endless *data fountain*. Similar to RS codes, they have the property that the original data can be decoded from any $\hat{k}$ chunks, with $\hat{k}$ being only slightly larger than k. Especially for large k, there exist fountain codes with small encoding and decoding complexities [221].

In the next section, we consider how packets can be encoded into data chunks by using (i) an RS code, (ii) a random linear fountain code (RLF), and (iii) a Raptor code. The data dissemination can then be performed in several ways. Section 3.4.2 describes two simple protocols that perform the distribution of bulk data throughout a network. Data chunks can be sent either by means of acknowledgements and retransmissions or by employing a request-based approach, where a receiver requests missed chunks. Both protocols may take advantage of FEC to reduce the number of data transmissions. Real-world evaluation results are presented in Section 3.4.3.

[8] Therefore, the erasure positions must be known. Otherwise, $k + \frac{n-k}{2}$ symbols must be received.

3.4.1 FEC Schemes for Bulk Data Dissemination

Reed-Solomon Codes

As described in Section 3.3.1, an RS(n, k) code encodes k data symbols into n symbols of size m with redundancy $n - k$ per codeword. If only erasures occur, an RS decoder is able to decode a codeword if it receives k arbitrary symbols from the codeword. In contrast to Section 3.3, where RS codes are applied to individual packets in order to account for bit errors [171], it is also possible to spread the code over several data chunks.

Nonnenmacher *et al.* [184] proposed the use of RS codes for reliable multicast transmissions, but the idea can also be applied to data dissemination in general. For example, consider the data block of size lk depicted in Figure 3.19 that is divided into k chunks $c_1 \dots c_k$ consisting of l symbols each. The symbol size m is assumed to be 8 bits. Data chunks are encoded by using l parallel RS(n, k) encoders that generate *parity* chunks $p_1 \dots p_{n-k}$ carrying redundant information. Therefore, k symbols at the same position i within a chunk are considered in order to generate the appropriate parity symbols. Interleaving all parity symbols finally leads to $n-k$ parity chunks of the same size. It should be noted that even if the data block consists of only $l\tilde{k}$ bytes with $\tilde{k} < k$, the same RS(n, k) encoders can be applied by using $k - \tilde{k}$ padding chunks. If the encoding as well as the decoding side are aware of $\tilde{k}$, these padding chunks need not be transmitted as the content is known *a priori*[9].

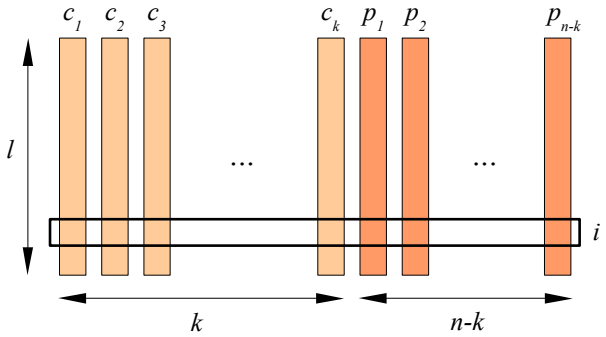


Figure 3.19: An RS-encoded data block

After the encoding process, a sender that would like to transmit the data block to one or several receivers initially sends the original data chunks $c_1 \dots c_k$ only. If then a receiver misses, e. g., d chunks during the transmission, parity chunks will be transmitted in addition. As long as d is smaller than (or equal to) the number of parity chunks, sending only parity chunks should be sufficient. Otherwise, it will also be necessary to resend data chunks, in order to allow the receiver to decode the complete data block. It is an important question which of the original chunks should be resent, without sending too much redundancy. One solution would be to restart the data stream with the first encoded chunk. However, it is likely that those chunks will be redundant for other nodes that have received them before. Especially in a multi-hop network where data is forwarded by several nodes, always starting with the first chunk may thus be very inefficient. In contrast, sending parity chunks has the advantage that it is not the *same* packet that is resent.

[9] Padding chunks just consist of a long run of zeros.

Hence, another solution would be to generate a random sequence of n chunks, which are broadcast on request. In addition, requesting nodes could specify which chunks they have already received, using unique chunk identifiers. As it is expected that this strategy will achieve a better performance, we will use it in the evaluations presented in Section 3.4.3. In the following we consider how the problem can be tackled by fountain codes.

Random Linear Fountain Codes

Fountain codes were first mentioned by Byers *et al.* in [45], but without an explicit construction. The idea is to produce an endless stream of output symbols of equal size for a set of k input symbols. Since the symbol size does not influence the code itself, a symbol can also be a complete chunk packet. To decode and recover the k original symbols, it is then sufficient (with high probability) to receive any set of n output symbols[10]. Good fountain codes require n to be only slightly larger than k and are able to achieve a decoding complexity that is linear in k. The first instance of such a code was invented by Luby [158, 159] and called LT code. However, as LT codes perform worse for small k, they need to be extended by an additional *outer code*. The combination is called a Raptor code, which we describe later.

Despite a higher complexity, the simplest form of fountain codes are *random linear fountain* (RLF) codes, which have nearly optimal properties concerning the Shannon limit[11] [219]. Given a set of k chunks $c_1 \dots c_k$, an RLF encoder randomly combines $\hat{k}$ chunks with $\hat{k} \leq k$ to one output chunk $\hat{c}_i$ that is sent to one or several receivers afterwards. Combining two or more chunks is done by an exclusive-or operation as shown in Figure 3.20.

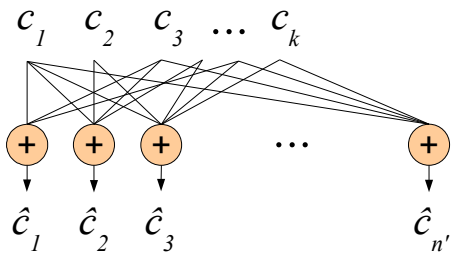


Figure 3.20: RLF encoding is done by combining original chunks randomly

Which chunks are to be combined is determined by the generation matrix G. The encoder generates a bit vector of k random bits for each output chunk $\hat{c}_i$ stored at row i in G. The transmitted output chunk is then generated as

$$\hat{c}_i = \sum_{j=1}^{k} c_j G_{ij}^T. \tag{3.25}$$

[10] In the following, we rather refer to chunks instead of symbols.

[11] Shannon's theorem gives an upper bound on the maximum amount of error-free digital data that can be transmitted over a communication link with a specified bandwidth in the presence of the noise interference [255].

Figure 3.21 illustrates an example of such a binary generation matrix G with k columns and n' ever-growing rows. By transmitting the bit vectors or by using a pseudo-random generator where the key seed is known to the sender as well as to the receivers, the same generation matrix $\hat{G}$ can be constructed, as shown on the right-hand side of Figure 3.21 from the receiver's point of view. Note that some chunks are assumed to get lost during the transmission, indicated by red rows. Thus, $\hat{G}$ may be of a different size than G.

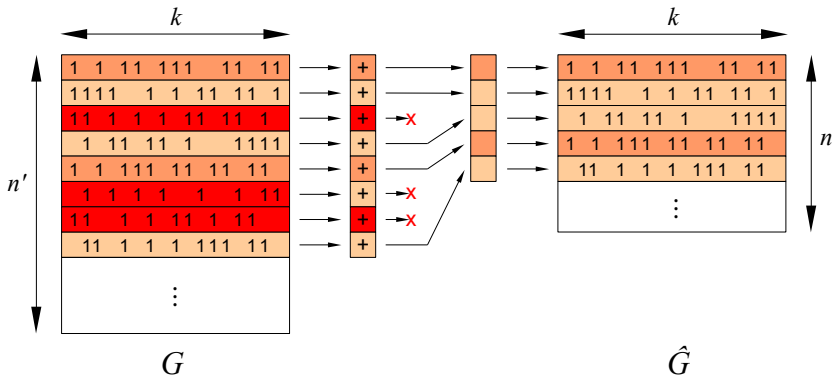


Figure 3.21: Illustration of the RLF encoding and decoding generation matrix

Decoding the received chunks $\hat{c}_1 \dots \hat{c}_n$ is only possible for $n \geq k$. Otherwise, the receiver has not received enough chunks. If $n = k$ and $\hat{G}$ is invertible (modulo 2), $\hat{G}^{-1}$ can be computed by, e. g., Gaussian elimination[12]. In this case, the original chunks $c_1 \dots c_k$ can be computed from

$$c_i = \sum_{j=1}^{k} \hat{c}_j \hat{G}^{-1}_{ij}. \tag{3.26}$$

$\hat{G}$ is invertible if and only if it contains k linearly independent rows or columns, respectively. Linearly dependent vectors identify chunks that are redundant. Thus, at the receiving side, the decoder skips all chunks that show a linear dependency on already received bit vectors until $n = k$. Then, the generation matrix $\hat{G}$ can be inverted, and the decoder will be able to recover the original chunks $c_1 \dots c_k$.

Unlike RS(n, k) codes, fountain codes are independent of the bit error rate on the wireless channel. Since the encoder generates an endless output stream, the receiver is eventually able to decode all received chunks. It may just take more time if the error rate increases. However, in order to stop the fountain, a receiver must somehow indicate that the decoding was successful.

It might be interesting to know how many excess (redundant) chunks $n - k$ a decoder will receive on average even if no packet loss occurs. That is, what is the probability to receive e excess chunks? MacKay shows in [163] that the probability $p(e)$ that a receiver will not be able to start decoding after e excess chunks have been received is bounded by

$$p(e) \leq 2^{-e} \tag{3.27}$$

[12] In practice, often an LU decomposition algorithm is used.

for any k. Thus, the probability that a receiver will need more than six excess chunks is less than one percent. On the other hand, the probability that a receiver will be able to start decoding after k chunks ($e = 0$) is only 0.289, even if an optimal RLF code is used[13]. Figure 3.22 depicts the upper bound function 2^{-e} and the real probability function $p(e)$ obtained by means of simulations and real-world evaluations. For $e > 1$, the upper bound 2^{-e} is quite tight as $p(e)$ is very close to it. Furthermore, we can see that the RLF implementation on the ESB nodes performs very well and leads to the same results we obtained by simulations.

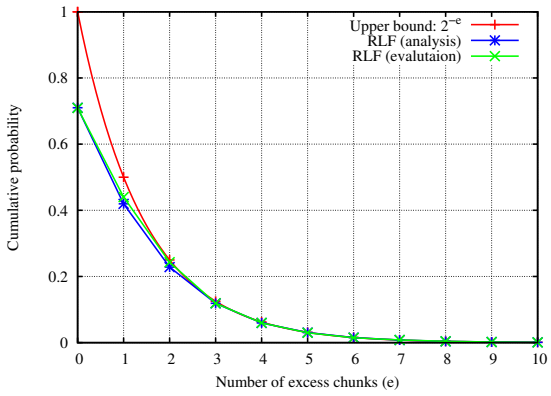


Figure 3.22: Probability distribution of RLF decoding failures

Raptor Codes

Although random linear fountain codes can get arbitrarily close to the Shannon limit, they have the disadvantage that their decoding costs are cubic, due to the calculation of $\hat{G}^{-1}$. As long as k is small, this might not be an important issue. However, for large k a more suitable solution would be preferable.

In [158], Luby presents an LT code which has an almost linear decoding complexity. The idea is to minimize the *density* (the number of ones) in the generation matrix G. Low-density codes have the advantage that decoding can easily be performed by using an algorithm called *belief propagation* [187], which is also known as the sum-product algorithm. Luby proposes a special probability distribution from which the density of outgoing bit vectors is chosen. The design of this distribution is critical since it heavily affects the quality of the code. The density for most chunks must be low in order to start the decoder. But occasionally, the density must be high enough to ensure that every original chunk will be covered by outgoing chunks. Otherwise, it will not be possible to decode the complete data block, as uncovered chunks cannot be reconstructed.

Raptor (RT) codes [221] extend LT codes by an additional *outer* code, aiming to minimize the probability that uncovered chunks will occur. The *inner* code is an arbitrary LT code with an average density of about three. This ensures that the decoder will not be blocked and can start. However, due to the low density, it is unlikely that all original chunks will be covered by the inner code, keeping receivers from decoding the chunk block. Thus, the idea is to use the outer code to recover such uncovered chunks

[13] Optimal means that the generation matrix is created by an optimal random number generator.

by FEC. Good outer codes are, for example, irregular low-density parity-check codes [202, 203]. But any other code can be used, too.

3.4.2 Data Dissemination Protocols

In order to evaluate the presented FEC coding schemes in a WSN, we need an appropriate data dissemination protocol. Since in a real WSN most of the nodes will not be in the vicinity of the sink node, the protocol must account for multi-hop dissemination. In this section, we propose two simple and distributed protocols for which we evaluate the above-mentioned FEC codes. The first protocol is based on reliable unicast communication. Packets are acknowledged and if necessary retransmitted by the sender automatically. In contrast, the second protocol avoids the need of acknowledgements by using a request-based approach. Available data is first announced to neighbors and then requested by interested nodes only. Should some data packets get lost during the transmissions, retransmissions will be triggered by requesting lost packets.

Acknowledgement-Based Data Dissemination

The acknowledgement-based data dissemination protocol is similar to traditional unicast communication. Data is addressed and sent to one receiver, which acknowledges each chunk received correctly. In so doing, a receiver should get the complete set of data chunks since chunks which have not been acknowledged are resent. In such a case, it will depend on the encoding scheme whether the same chunk or a different one is retransmitted.

Although the communication is unicast concerning its addressing, note that the physical transmission is broadcast due to the wireless medium. Thus, other receivers within transmission range are able to receive chunks in *promiscuous mode*. Hence, after the actual receiver has obtained all data chunks, it is likely that adjacent receivers will have received most of the chunks by overhearing, too.

The complete acknowledgement-based protocol works as follows: First, a sending node announces data by broadcasting an arbitrary number of data chunks. Each chunk contains an identifier that is generated by the original data source. In addition, it includes both a sequence number and the total number of chunks required to reconstruct the original data. Nodes interested in the data answer with an acknowledgement, indicating the next chunk they need. In order to avoid collisions, acknowledgements are sent after expiry of a random backoff time. Once an acknowledgement has been received, the source node will start sending data chunks to the appropriate node until all chunks have been acknowledged. Other nodes interested in the same data will try to overhear the chunks and keep track of missed and erroneous ones. After the source node has stopped sending and the channel becomes free, nodes needing further chunks to complete will again set a random backoff timer. As soon as the timer expires, an additional acknowledgement will be sent that indicates the loss of chunks.

If all receivers are within the source's transmission range, no further overhead is required. However, in order to account for multi-hop networks, it is necessary that additional nodes act as data sources to cover distant nodes, too. Thus, as soon as a node has recovered the data block completely and the

medium becomes free, it will broadcast some chunks itself. Again, random backoff timers are used to avoid collisions. However, since we cannot assume that the network topology is known, *each* node will try to find uncovered neighbors, even if this causes much redundancy.

Request-Based Data Dissemination

An advantage of the acknowledgement-based protocol is that data chunks will be resent automatically if they could not be received successfully. No additional communication overhead is required in this case. However, the overhead due to acknowledgements is quite high, especially if only a few packets get lost.

Thus another idea is to use a request-based protocol. Like before, the data source will start by broadcasting some chunks in order to find interested nodes. But instead of sending acknowledgments, a *request* will be sent back that contains a list of chunk IDs needed by the appropriate node[14]. In addition, each node that has discovered a new data stream will set a request timer for the case that requests get lost. Once the source node has received a request, it will start broadcasting the requested chunks without any further communication overhead. Afterwards, it will wait for additional requests that may restart the data stream. Since each node will hold a pending request timer until it has received all chunks correctly, missed chunks will either be requested by the same node or, possibly later on, by others.

Concerning multi-hop networks, the protocol works like the acknowledgement-based one. Thus, each node that has received the complete data block will act as an additional data source. Upon initially broadcasting some data chunks, uncovered nodes should then be able to request missing chunks. But in contrast to the acknowledgement-based protocol, requests are not sent by unicast but broadcast, which allows any data source in the neighborhood to answer.

3.4.3 Experimental Evaluation

We have evaluated the acknowledgement-based data dissemination protocol (using up to ten retransmissions), as well as the request-based protocol in a real sensor network with raw transmissions (non-coding), an RLF code, an RS$(255, 223)$ code, and a Raptor code. As the request-based protocol is expected to cause a smaller overhead, it is used in combination with FEC. The differences between the acknowledgement-based protocol and request-based protocol are illustrated by means of non-coding only, as they were analogous for the FEC codes.

Single-Hop Evaluation

At first, we consider a single-hop environment where six ESB nodes are placed within the communication range of one source node. The source node periodically generates data chunks that are broadcast

[14] Depending on the coding scheme, it may not be necessary to include a complete list but only the number of requested chunks.

to all other nodes. The complete data block consists of 32 chunks (k) which have a size of 64 bytes. To obtain stable results for different performance metrics, we repeated each experiment 250 times.

As we are interested in single-hop effects, the source node sends data chunks only. In the next section, we also analyze the effects if packets are forwarded in a multi-hop fashion through a larger network. Furthermore, in order to maintain the single-hop environment, all nodes use their maximum transmission power. Packet loss is produced artificially according to a uniformly distributed loss rate by dropping chunks randomly after reception. In order to analyze the influence of packet losses, the loss rate is varied between 0 and 0.9 during the experiment.

The evaluation results obtained are presented in Figure 3.23. Figure 3.23(a) shows the number of chunks that must be sent by the source node until all nodes have received the complete data block. Thus, it can be used as an indicator of the channel utilization and the energy consumption. If packets get lost, the best results will be achieved by the RS and RLF codes, which both require up to 40% fewer chunks to be sent than does non-coding. Hence, encoding performs very well because redundantly received chunks can be used to recover missed ones (instead of requesting them again). However, the RT code performs worse since its decoding algorithm requires more chunks in order to start and complete than does the RLF code. Only for packet loss rates larger than 0.25, does it also outperform non-coding.

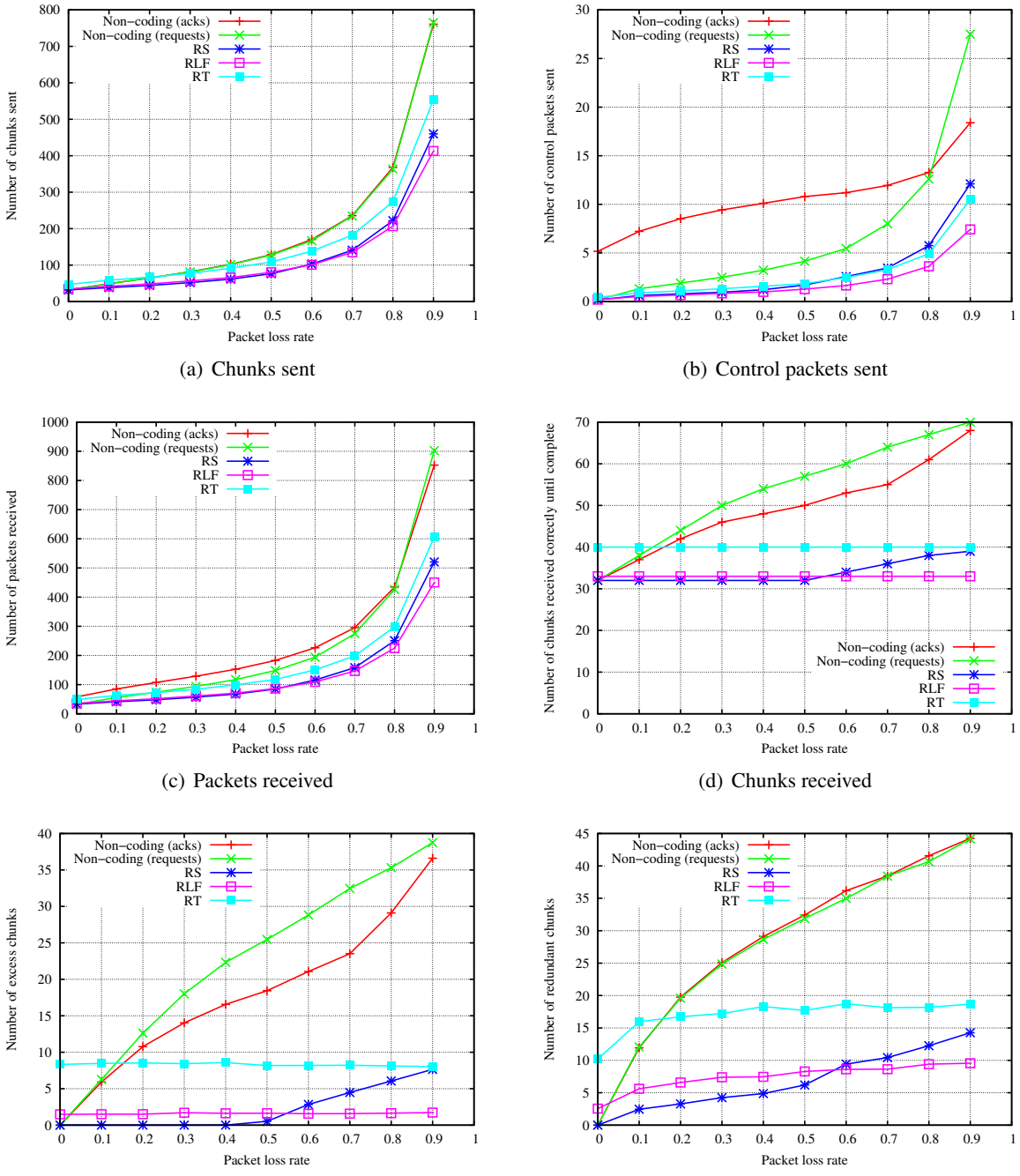
Also illustrated in Figure 3.23(a), using acknowledgements or requests makes no difference concerning the number of sent chunks. Because the number of missed chunks is in both cases nearly the same, the only difference is the manner in which chunks are resent. This can be done either by actively requesting them or by automatic retransmissions.

The average number of sent control packets (acknowledgements and requests) is shown in Figure 3.23(b). We see that using acknowledgements generally causes more overhead, except for very high loss rates. That is due to the fact that each chunk needs to be acknowledged. If the loss rate is low, it is therefore more efficient to request missing chunks at once. However, if the loss rate increases, chunks that are still missed will cause many requests until they have been resent. In contrast, the acknowledgement-based protocol resends chunks automatically if no acknowledgement is received. Thus, the increase in the number of sent requests is significantly higher for non-coding. If FEC is employed, fewer control packets will be required because missed chunks can often be recovered by other chunks.

Figure 3.23(c) depicts the total number of received and overheard packets[15], including chunks and control packets. As fewer chunks will need to be sent if FEC is used, the number of received packets is significantly lower than for non-coding. The difference in the acknowledgement- and request-based protocols is due solely to the number of control packets used. Up to a loss rate of about 0.8, using acknowledgements causes more overhead. However, for higher loss rates the number of requests dominates, preferring automatic retransmissions as used by the acknowledgement-based protocol.

The average number of chunks that have to be received chunks before the data block is complete is depicted in Figure 3.23(d). In addition, Figure 3.23(e) shows the the number of excess chunks, i. e.,

[15]Packets that are received either by unicast, broadcast, or in promiscuous mode.



(a) Chunks sent

(b) Control packets sent

(c) Packets received

(d) Chunks received

(e) Required excess chunks

(f) Redundant chunks

Figure 3.23: Evaluation results from the single-hop experiment ($k = 32$)

the number of chunks necessary before the decoding can start. In the case of non-coding, it shows the number of chunks until the last missing chunk has been received. Thus, both figures illustrate how fast an application can process the data block and how long a node must listen to ongoing transmissions on the wireless channel.

While the number of excess chunks is almost constant for RLF and RT coding, the RS code requires more chunks in order to start its decoding process if the number of lost packets grows. Its sudden

increase of excess chunks can be explained as follows: As the RS encoded data block consists of 64 *different* chunks (32 original data chunks plus 32 redundant chunks)[16], multiple nodes may benefit from transmissions as long as not all chunks are sent. However, retransmissions will be required if the packet loss rate is larger than 50% because then the expected number of correctly received chunks will be less than 32. But since *resending* the same chunks may be beneficial to only a few nodes, it will cause a higher redundancy for other nodes. The threshold of 50% can be calculated with $1 - k/n$, where k denotes the number of original chunks and $n - k$ the number of redundant chunks. Hence, if the data block consists of more than 32 data chunks, the threshold will even decrease, favoring the use of RLF codes as shown in Figure 3.23(d) and 3.23(e).

Although using either acknowledgements or requests makes no difference concerning the number of transmitted chunks, the average number of excess chunks can be reduced if automatic retransmissions are employed. The acknowledgement-based protocol benefits from sending chunks to the same node until this node has received all chunks completely. This minimizes the *average* number of excess chunks significantly, as such nodes need no further packets for decoding. However, this advantage vanishes if the maximum number of retransmissions is not sufficient. In this case, the acknowledgement-based protocol behaves similarly to the request-based one. Thus, both protocols show almost the same performance for very high packet loss rates if FEC is not applied.

In addition to the number of excess chunks, Figure 3.23(f) shows the average number of chunks received redundantly, i. e., chunks which have already been received or which contain no new information for the decoding process. As long as no packet losses occur, neither non-coding nor RS coding will cause either excess or redundant chunks. In contrast, the RLF and the RT codes will always lead to some redundancy due to their encoding features of combining data chunks randomly. Again, the RS code shows the best performance as long as chunks need not be retransmitted. Thus, for the loss rates greater than 0.5, it is outperformed by RLF coding. The redundancy of non-coding is significantly higher as a consequence of more chunk packets being sent. But since the acknowledgement- and request-based non-coding protocols eventually send almost the same number of chucks, the redundancy of both protocols does not differ much.

In conclusion, employing FEC codes reduces the channel utilization and the number of redundantly transmitted packets in sensor networks significantly. The best performance is achieved by the RS code if the packet loss rate is moderate and the data block consists of only a few chunks. Higher loss rates favor the use of RLF coding. Although the RT code still outperforms non-coding, it causes high overhead. Many chunks need to be created until its decoding process starts. Thus, it is not very suitable for disseminating data as long as the data block is not considerably larger.

Multi-Hop Evaluation

Whether or not the previous results can be verified also in a multi-hop network is an issue addressed in this section. We installed a testbed consisting of 25 ESB nodes in our lab. As shown in Figure 3.24, nodes are placed in a 4×6 grid structure, using an additional source node at the bottom of the grid.

[16] Note that an RS$(255, 223)$ code is used.

The distance between two nodes is 60 cm. In order to provide long-time evaluations, each node is connected to an external power supply. Furthermore, an eGate/WEB device is used to access the network over the Internet, allowing the flashing of arbitrary applications and protocols, as well as the control and monitoring of the network. The source/sink node is connected to a notebook that stores log information in a database for later processing.

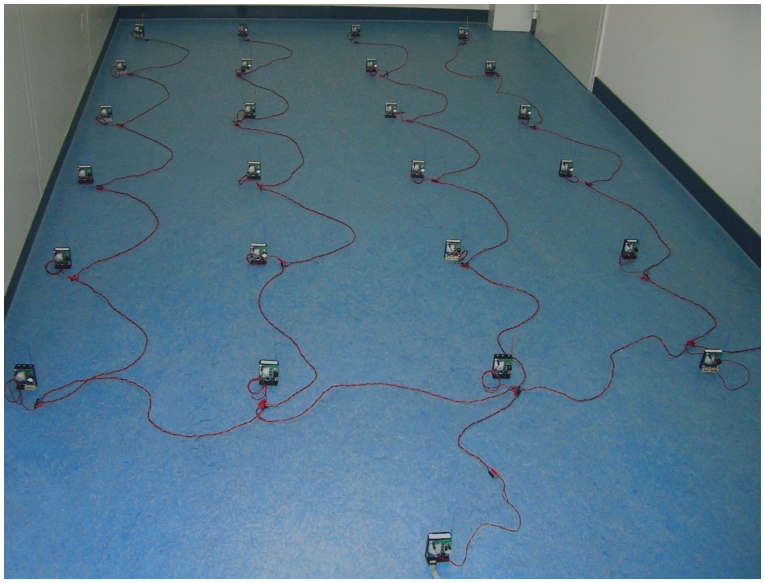
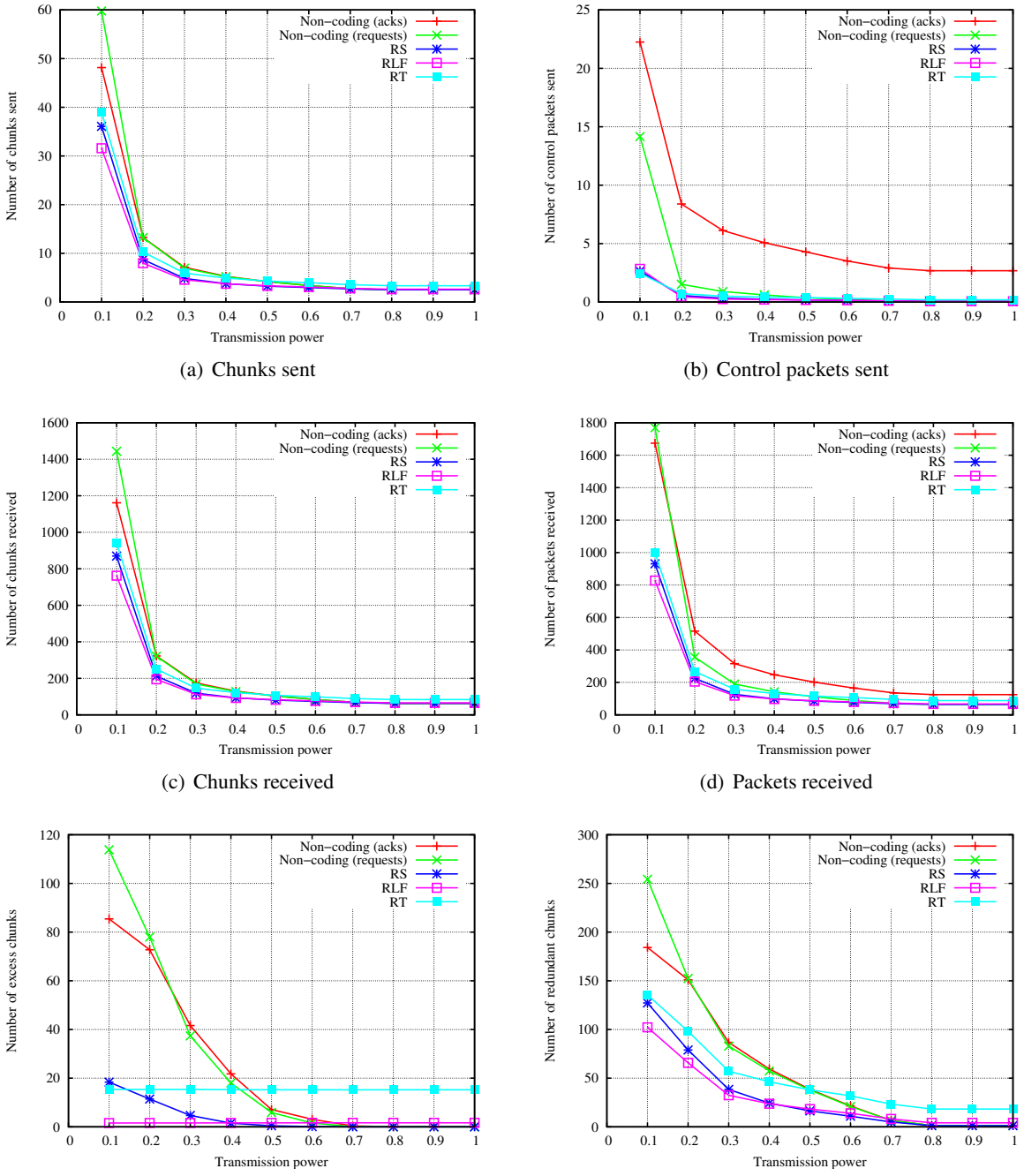


Figure 3.24: Wireless sensor network testbed

In this experimental evaluation, the source node disseminates a data block consisting of 64 chunks by employing the acknowledgement- and request-based protocols for non-coding as well as for RS, RLF, and RT coding. The chunk size is 64 bytes. In order to evaluate the effect multi-hop forwarding has on network performance, the nodes' transmission powers are varied from 0.1 to 1 successively. Lower transmission powers were not considered because they often caused partitioned networks.

The evaluation results are depicted in Figure 3.25. Figure 3.25(a) shows the number of sent chunks averaged over all nodes in the network. While the transmission power is high, few packet losses occur, and most chunks are received directly from the source. In this case, encoding data shows no benefit over non-coding. It even performs worse since the computational overhead is not negligible. However, for low transmission powers, the loss rate in the network increases, and data chunks will need to be forwarded more often. For a data block consisting of 64 chunks, RLF coding performs best. The RS code sends more chunks because its code redundancy is fixed *a priori* and not randomly generated like in the fountain codes. Thus, as we have discussed before, it suffers from higher loss rates as soon as all parity chunks are sent, and data chunks must be retransmitted.

Concerning the difference between using requests or acknowledgements, the acknowledgement-based protocol shows a better performance for low transmission powers. This can be explained by the non-negligible number of asymmetric links, which will be even higher if the transmission power decreases. Compared to Section 3.4.3, where only chunks were artificially dropped, we now must also deal with lost control packets. For example, consider an asymmetric link between two nodes where the forward link is highly lossy. If a node receives a request over this link, the request-based protocol will start



(a) Chunks sent

(b) Control packets sent

(c) Chunks received

(d) Packets received

(e) Required excess chunks

(f) Redundant chunks

Figure 3.25: Evaluation results from the multi-hop experiment ($k = 64$)

sending the requested chunks (in this case, 64 chunks). However, it is likely that almost all chunks will get lost, making the entire transmission useless. In contrast, the acknowledgement-based protocol stops after ten retransmissions. Thus, if the number of requested chunks is significantly higher than the maximum number of retransmissions, the acknowledgement-based protocol is preferable.

The average number of sent control packets (request and acknowledgements) is shown in Figure 3.25(b). While all FEC codes lead to a similar overhead for control packets, non-coding generates

up to five times more request packets. The high number of acknowledgements increases the overhead even more. In Figure 3.25(c), the average number of received, respectively overheard chunks, is depicted. If control packets are also considered, Figure 3.25(d) illustrates the total number of received packets. RLF coding again shows the best performance, followed by RS, and RT coding. Regarding the number of received chunks that contain no error and can thus be processed by an FEC code, Figure 3.25(e) shows the average number of excess chunks. Except for non-coding, the results are quite similar to the ones shown in Figure 3.23(e). However, due to asymmetric links, the acknowledgement-based protocol causes more excess chunks because of automatic retransmissions. Nearby nodes may thus receive chunks more than once until they have received the entire data block. For a transmission power of 10%, these effects vanish because the packet reception rate substantially deteriorates, decreasing the fraction of nodes affected by retransmissions.

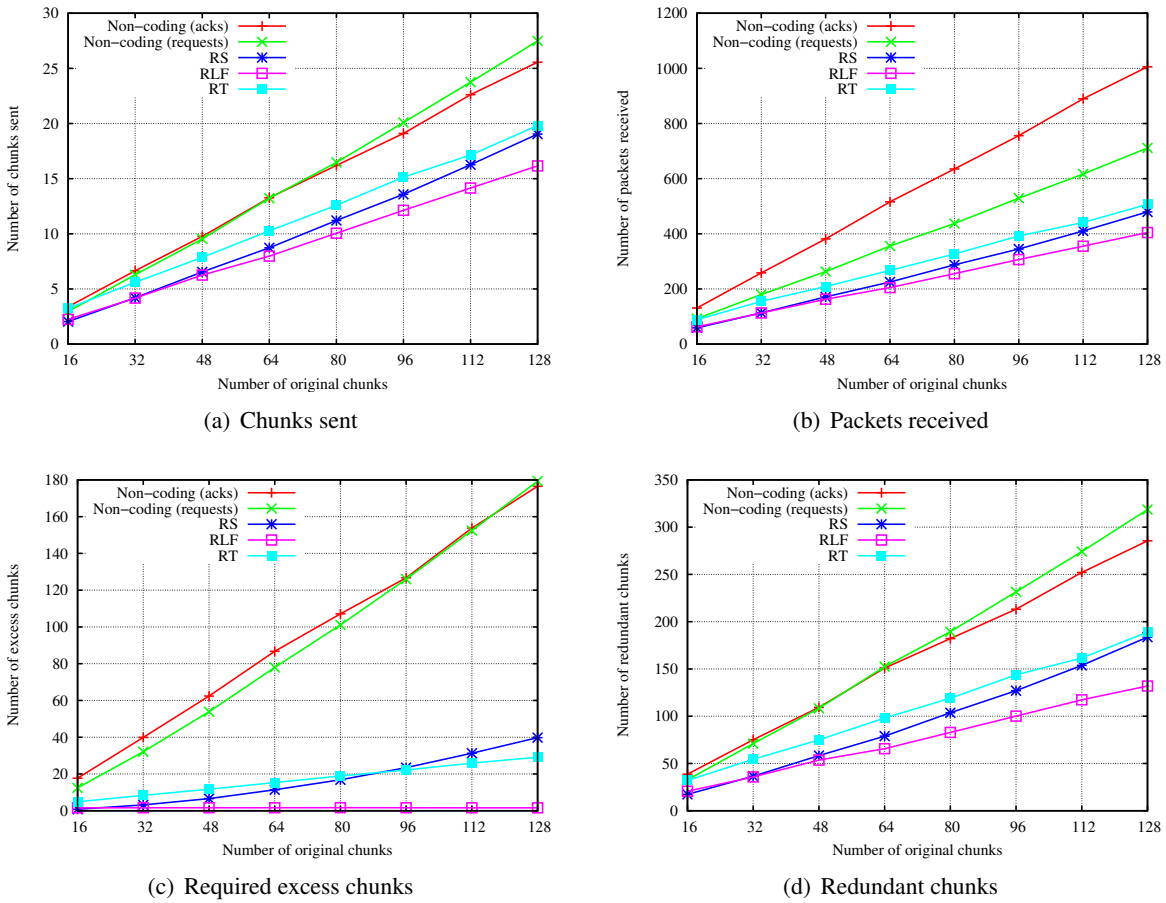
Finally, Figure 3.25(f) considers the average number of chunks which has been received correctly but redundantly. As the acknowledgement-based protocol sends fewer chunks for low transmission powers, the number of redundantly received chunks will also be lower. Thus, in this case, it outperforms the request-based protocol. If the majority of links do not experience a perfect reception, employing FEC codes is particularly recommended. However, it is still an open question whether to use RLF or RS coding, since the performance heavily depends on the loss rate and the number of chunks in the data block.

We have thus analyzed the influence of the data block size in more detail, using a transmission power of 20%. Figure 3.26 shows the results for the average number of chunks sent, the number of packets received (including control packets), and the number of excess and redundant chunks. We can clearly see that the larger the data block size, the worse the performance of RS coding. Although the RS code is still better than non-coding, it is significantly outperformed by RLF coding. As before, the overhead of the RT code is considerable. Even if the data block consists of 128 chunks, RS and RLF coding perform better. Regarding the difference between the request- and acknowledgement-based protocols, Figure 3.26(a) indicates that the acknowledgement-based protocol will cause fewer chunk transmissions if the data block size is larger than 64 bytes[17]. However, the overhead for acknowledging each chunk received becomes substantial if the block size increases as illustrated in Figure 3.26(b). It thus seems that a hybrid protocol may be the best solution. By limiting the number of chunks which can be requested at once, the trade-off between the overhead due to control packets and useless chunk transmissions due to asymmetric links could be taken into account. In this way, the advantages of both protocols could be combined, improving the performance of the request-based protocols in the presence of asymmetric links.

3.4.4 Conclusions

Using FEC codes for the dissemination of bulk data helps to reduce the number of redundant packets significantly, especially in lossy environments. Among the evaluated FEC codes, RLF and RS coding show the best performance. For low packet loss rates, the RS code needs less redundancy and thus performs slightly better. However, if the loss rate increases, it will suffer from retransmitting identical

[17] Note that if more than ten retransmissions are used, the threshold would be higher.



(a) Chunks sent

(b) Packets received

(c) Required excess chunks

(d) Redundant chunks

Figure 3.26: Influence of the data block size in the multi-hop experiment ($txp = 0.2$)

chunks as soon as the predefined code redundancy has been exhausted. This disadvantage is tackled by the RLF and RT codes. Both codes generate output chunks by randomly combining original data chunks. Thus, it becomes unlikely that identical chunks will be sent. However, although the RT code has a better decoding complexity, it is clearly outperformed by RLF coding, which is mainly due to the relatively small number of data chunks we considered.

Hence, it may be advisable to use an optimal RLF code in conjunction with a request-based dissemination protocol if the number of requested chunks is bounded, e. g., to ten packets. As long as the size of the data block is not too large, its decoding complexity can even be handled by low-power hardware with limited processing capacities. RS codes are better suited to the correction of bit errors within a packet, as was investigated in Section 3.3. Combined with the resync mechanism presented in the first part of this chapter, we now have a comprehensive framework for the efficient dissemination of data to multiple receivers in a wireless sensor network.

The next three chapters will deal with the problem of how the nodes themselves can transmit sensed data or any other kind of requested information to a predefined sink. Due to the energy constraints of the network, this requires approaches that mainly take energy efficiency into account. Although we will not explicitly consider the impact of resync and FEC, both techniques can easily be integrated as they are complementary to the protocols presented in following chapters.

CHAPTER 4

Energy-Efficient Forwarding

"Genius is 1 percent inspiration and 99 percent perspiration. As a result, genius is often a talented person who has simply done all of his homework."

– T. Edison –

4.1 Introduction

Many recent experimental studies have shown that, especially in the field of sensor networks where low-power radio transmission is employed, wireless communication is far from being perfect [48, 72, 265, 276]. Thus, widely used communication models corresponding to binary links with either full connectivity or none at all are not realistic. Instead of modeling only a connected and a disconnected region as in the *unit disk graph* model, more realistic loss models consider a transitional region with a widely varying degree of packet loss [51, 258, 282]. Although for a sender-receiver pair the packet reception tends to decrease with growing distance, there might be some cases where more distant nodes have smaller loss rates than do nearby ones. Thus, exploiting nodes located in the transitional region might improve the efficiency of a forwarding strategy significantly [64, 217]. In terms of energy, it might be more efficient to establish longer paths that experience few packet losses instead of shorter ones that cause many transmissions until packets have reached their destinations.

In this chapter, we will explore the efficiency of different forwarding strategies, which can be used by sensor nodes to report data to a predefined network sink. For such many-to-one communication, many routing algorithms rely on distance-based forwarding, where the number of hops serves as a distance metric [124, 126, 190, 212, 266]. However, since the energy consumption and the connectivity between nodes depends on the link quality, it is not obvious which neighbor should forward packets in order to be energy-efficient. If each node simply selects the node with the lowest hop counter, it is likely that the end-to-end path will exhibit high packet losses, leading to a poor efficiency due to many retransmissions.

Several experimental studies have explored this problem with different routing schemes [64, 217, 258, 265]. While most of the existing work focuses either on minimizing the expected number of transmissions or tries to maximize end-to-end delivery, we concentrate on *energy efficiency* in order to trade off packet delivery ratios and energy consumption. By means of mathematical analyses, simulations, and an implementation, we investigate a broad framework of distance-vector-based forwarding strategies for static wireless sensor networks. Furthermore, we propose two new forwarding schemes, namely *single-link energy-efficient forwarding* and *multi-link energy-efficient forwarding*.

In the next section, we first outline related work. Section 4.3 then describes our packet reception and energy model that is motivated by experimental studies; it is later used in the simulations. This section also gives information about assumptions made and the performance metrics we focus on. Section 4.4 provides an analysis of two simple forwarding schemes which are based on the path length counted in hops and on the packet reception ratio on a link. By blacklisting bad nodes, we explore to what degree such strategies can be improved. In Section 4.5, we present the single-link and multi-link energy-efficient forwarding strategies. Considering the case of infinite and finite retransmissions, we present a mathematical analysis of both strategies. By means of simulations, we compare the performance of all forwarding strategies in Section 4.6. Results from real-world experiments are presented in Section 4.7. Section 4.8 concludes the chapter.

4.2 Related Work

Several routing protocols for sensor networks concentrate on energy-related issues, which surely are important and challenging aspects [11, 67]. For many-to-one communication with multiple data-reporting nodes and one sink node, protocols like *directed diffusion* [124] use distance-vector-based routing. In the directed diffusion approach, the sink node first propagates an *interest* or *advertisement* throughout the network. By assigning a hop counter to each interested node, reverse paths are established by setting up *gradients* pointing to the neighbor with the lowest hop counter. The reverse paths then form a routing tree which is rooted at the sink and can be used for forwarding data reports. In addition to hop counters, other forwarding metrics, which can be defined by means of gradients, are also possible.

The initialization phase of establishing reverse paths is also used in many other protocols [108, 126, 190, 266]. In order to avoid burning out the nodes' energy along the shortest path, many approaches attempt to improve energy balancing among forwarding nodes [87, 212, 218]. *Gradient-based routing* (GBR) [212] improves directed diffusion by uniformly balancing traffic throughout the network, using data aggregation and traffic spreading to do so. Shah *et al.* [218] propose an energy-aware routing scheme that employs a probability function based on the energy consumption of different routing paths. Other energy-aware routing schemes are analyzed by Gan *et al.* in [87].

However, experimental studies have shown that packet loss is not uniformly distributed over distance; losses also occur for nearby nodes, and a significant number of links are asymmetric [48, 258, 276, 278, 282]. Due to this, simulation results of many routing protocols could not be verified in reality. In this chapter, we analyze the initialization phase of establishing reverse paths

for many-to-one communication in consideration of the packet delivery ratio and the energy efficiency for such lossy environments. Techniques for energy balancing throughout the network are orthogonal and considered in Chapter 5.

There are many attempts to improve the fault tolerance through robustness by multi-path routing [90, 227, 266]. Ganesan *et al.* [90] propose partially disjointed multi-path routing schemes, which they call *braided multi-path routing*. Compared to completely disjointed multi-path routing, they study the trade-off between energy consumption and robustness. In terms of energy efficiency, braided multi-path routing seems to be a viable alternative for recovering from node failures. Srinivasan *et al.* [227] address the problem of optimal rate allocation for energy-efficient multi-path routing. They propose a flow control algorithm which can be easily implemented and which provides the optimal source rates in a distributed manner. In [266], Ye *et al.* present *gradient broadcast* (GRAB), where packets travel towards the sink by descending a cost path. Costs are defined as the minimum energy overhead required to forward packets to the sink along a previously established path. Nodes close to the sink will have lower costs than will far-away ones. All nodes receiving a packet with a lower cost will participate in packet forwarding. Since multiple paths with decreasing costs exist, GRAB is quite robust and reliable with respect to the delivery of data. However, multi-path forwarding comes at the expense of a high energy consumption. In contrast, we tackle robustness by *multi-link forwarding*. Unlike addressing a specific node that is used to forward a packet, packets are broadcast to many potential forwarders[1], which we call a *forwarder set*. As it is more likely that one of these nodes will receive the packet correctly and thus be able to forward it, unnecessary retransmissions can be avoided. However, in order to conserve energy, multi-link forwarding tries to avoid multi-path forwarding and thus *actively* selects one node among all potential forwards only. Section 4.5 gives a detailed description of this multi-link forwarding.

In contrast to multi-path routing, Stann *et al.* [228] address the unreliability at the broadcast-level in low-power wireless networks of non-uniform density and propose *robust broadcast propagation* (RBP). RBP improves the reliability of flooding in a energy-efficient manner and trades off reliability and energy consumption very well. As flooding is often one of the building blocks for routing, reliable broadcast may be beneficial for many upper-layer protocols. In high-density neighborhoods, near-perfect flooding reliability will be achieved by moderate broadcast reliability. On the other hand, areas of sparse connectivity will be identified by important links that require guaranteed reliability. Through both experiments and simulations, Stann *et al.* demonstrate the advantages of their hybrid approach and compare the efficiency of RPB and with plain flooding.

Yarvis *et al.* [265] have also performed real-world experiments for large-scale sensor networks and evaluated the performance of *destination-sequenced distance-vector routing* (DSDV) in reality. They propose *quality-based* routing, a forwarding scheme that attempts to maximize the end-to-end delivery ratio by measuring reception ratios on each link. As a result, links experiencing poor reception qualities are avoided; thus the overall packet loss decreases. However, since maximizing the end-to-end delivery ratio tends to prefer shorter but high-quality links, the path length in hops increases significantly. Thus, more packet transmissions are needed, leading to a higher energy consumption.

[1] Note that due to the wireless medium, unicast and broadcast communication differ in their addressing only.

In [72], De Couto *et al.* analyze the throughput of minimum hop count distance-vector-based routing protocols and observe significant losses in the overall end-to-end packet delivery. Minimizing hop counters likely maximizes the geographically traveled distance per hop, which may degrade the packet delivery ratio per link. Thus, traditional routing protocols like *dynamic source routing* (DSR) and *destination-sequenced distance-vector routing* (DSDV) perform poorly. In [64], the authors extend their previous work and propose a minimum transmission metric that is approximated by $\frac{1}{forward\ link\ quality} \times \frac{1}{backward\ link\ quality}$, which now incorporates both packet reception ratios and link asymmetries.

Like De Couto *et al.*, Woo *et al.* focus on a *minimum transmission* (MT) metric [258]. They present experimental studies for forwarding schemes like shortest path forwarding, MT forwarding and techniques used in DSR and DSDV. However, minimizing the expected number of transmissions may not always result in the most energy-efficient forwarding path. For this reason, we rather focus on maximizing the energy efficiency, which can be defined as the delivery ratio per energy cost.

Similar to our energy efficiency metric, Cao *et al.* [47] optimize the energy per bit in the presence of unreliable radio links. The authors propose a joint-optimization process that regards both the recovery of lost packets on the link layer and the path selection on the routing layer. Also, Saukh *et al.* [209] focus on the end-to-end energy efficiency of routing paths. While their model accounts for different transmission powers, the existence of asymmetric links is not taken into account. In [61], Ciciriello *et al.* study the problem of efficient routing from multiple sources to multiple sinks. Based on periodic adaptation of routing paths, the routes that minimize the number of network links exploited are created. The work in [80] by Egorova-Förster and Murphy addresses the problem of multiple sinks in the context of single sources. Data is sent along *path sharing trees* in order to minimize the number of transmissions. During a learning phase, feedback from adjacent neighbors regarding the best routes found is gathered to explore alternative routes. Afterwards, discovered paths are used for routing.

An integration of MAC and routing features is presented in [205] by Ruzzelli *et al.* The work is tailored to minimize the high latency of packets that may occur if energy-efficient MAC protocols are used in combination with routing protocols. Packet forwarding is provided according to a multicast up- and downstream approach. The network is divided into timezones which are used to forward packet to the closest gateway in the network. By means of an appropriate scheduling policy, the latency of packets can be reduced significantly while still maintaining low duty cycles.

In [217], Seada *et al.* propose energy-efficient forwarding strategies for geographic routing by studying the effects of lossy environments. They focus on greedy forwarding, where each node tries to forward packets to nodes that are closest to the sink with respect to geographic distance. Such maximum-distance greedy forwarding techniques work well under ideal conditions but poorly in realistic environments. It turns out that many packets are transmitted on lossy links, leading to bad end-to-end delivery rates with high energy costs. Seada *et al.* therefore suggest a metric that is based on the product of the packet reception rate and distance. Their simulations have shown that this metric achieves optimal results by balancing longer, lossy links and shorter, high-quality links. Independent of this work, Zorzi and Armaroli have proposed the same link metric [279]. A more general framework is presented in [146], which uses a link metric called *normalized advance* for geographic routing. Similar to the work in [217, 279], packet forwarding is performed by neighbors which trade off optimally

between proximity and link cost. But in contrast, various types of link cost can used, e. g., packet error ratio, delay, or power consumption.

Apilo *et al.* [13] compare the performance of four local forwarding strategies for geographic routing. The forwarding methods differ in the selection of the next hop, which may be either greedy, random, weighted random, or opportunistic. While in greedy forwarding the traffic is concentrated on deterministic paths, random forwarding spreads the traffic more efficiently. However, the best performance has been achieved by opportunistic forwarding, in which the best *available* neighbor is chosen. While all these works rely on location-awareness, our work studies energy-efficient forwarding for non-geographic routing.

The concept of *opportunistic routing* [25, 26, 59, 277, 281] is closely related to our multi-link forwarding strategy. While traditional routing schemes try to find the best path and forward packets to a predefined, corresponding next hop, opportunistic routing takes advantage of the broadcast nature of wireless transmissions. Rather than best-path routing, the next hop is selected dynamically on a per-packet basis, depending on which nodes have received the packet successfully. ExOR is such an opportunistic scheme that was initially proposed by Biswas *et al.* in [25] and later extended in [26]. The general idea described in [25] is to forward packets to a predetermined candidate set; from among them the next hop is selected by means of priorities. The highest priority refers to the node that minimizes the *distance* to the destination most, while the distance may be defined by any routing metric desired. For example, ExOR estimates the *closeness* of nodes based on the expected number of required transmissions until a packet will reach its destination [72]. The selection of forwarding candidates and the priority scheme are the key challenges that need to be taken into account. In particular, duplicate transmissions must be avoided as they harm the performance gains of any opportunistic routing scheme. While the protocol proposed in [25] may suffer from such effects, the work in [26] describes a mechanism that eliminates duplicates successfully by means of robust acknowledgements. Packets are transferred in batches, while acknowledgements of higher priority candidates will be relayed by any other forwarding candidate. In so doing, retransmissions as well as duplicates are avoided in the majority of cases. However, due to the batch mechanism, ExOR is tailored only to single-flow bulk traffic. In addition, ExOR's routing metric does not take into account that packets are forwarded by candidates under opportunistic forwarding as well. Zhong and Nelakuditi [277] thus proposed an improved metric, which estimates the expected number of any-path transmissions under opportunistic routing. They also provide a comparison with deterministic routing, which was outperformed in terms of throughput. A comparison regarding a mobile wireless network is provided in [131].

In contrast to these works on opportunistic routing, we will rather focus on energy efficiency, taking into account the delivery ratio and the energy consumption of packet forwarding. While ExOR determines a comprehensive set of forwarding candidates, which may cause a high overhead as many nodes may be involved in forwarding, our multi-link concept will limit the number of forwarders to only three nodes. In so doing, only a small number of nodes need to receive a packet completely. Furthermore, our forwarding metric exploits multi-link forwarding *a priori* and provides an accurate estimate for the path's energy efficiency. While previous works did not account for the loss of acknowledgements [142, 281], we will take such effects into account explicitly. Packet duplicates are considered as well, and reduced by *actively selecting* the next forwarding node after the transmission of a packet. Even though this mechanism will cause a higher overhead and packet duplicates are still

possible, multi-link forwarding is often beneficial because the additional energy consumption will be considered in advance.

4.3 Models, Assumptions, and Metrics

4.3.1 Packet Reception Model

Many experimental studies have shown that the communication between a sender-receiver pair in a *wireless sensor network* (WSN) can be characterized as either a connected, transitional, or disconnected state [48, 51, 258, 276, 282]. Furthermore, it turned out that even if the distance between two sender-receiver pairs is the same, the reception characteristics are quite different. It was observed that the radio quality is affected by reflection, diffusion, multi-path effects, and ground attenuation [111, 133, 231]. Thus, especially in the transitional region, the reception of data might highly vary.

Motivated by these results, we model the *packet reception ratio* (PRR) on a wireless link according to an extended *log-normal shadowing model* [198]. Considering two arbitrary nodes a distance d apart from each other, this model can be intuitively described as follows: Below a distance D_1, nodes are almost fully connected, i. e., the packet reception ratio is mostly equal to one. In contrast, there is high probability that two nodes will be disconnected if they are more than a distance D_2 away from each other. In the transitional region between D_1 and D_2, the expected reception ratio decreases with growing distance but is influenced by some variation. The log-normal behavior within the transitional region can be modeled as

$$\widehat{prr}(d) = \begin{cases} 1 - \frac{1}{2}\left(\frac{d}{\delta}\right)^{2\alpha} & d < \delta, \\ \frac{1}{2}\left(2 - \frac{d}{\delta}\right)^{2\alpha} & \delta \leq d < 2\delta, \\ 0 & d \geq 2\delta, \end{cases} \tag{4.1}$$

for any arbitrary wireless link, with α being the attenuation exponent and δ being the average node's transmission range. In order to model different reception ratios within the transitional region, we introduce some variation that can be modeled by a Gaussian variable $X \sim N(0, \sigma(d))$, with $\sigma(d)$ being a distance-dependent standard deviation defined as

$$\sigma(d) = \beta \left(1.0 - \left|\frac{d - \frac{1}{2}(D_1 + D_2)}{D_1}\right|\right), \tag{4.2}$$

where β specifies the loss variation factor. Then, the packet reception ratio prr is calculated as

$$prr(d) = \begin{cases} [\widehat{prr}(d) + X]_0^1 & D_1 \leq d \leq D_2, \\ \widehat{prr}(d) & \text{otherwise} \end{cases}, \tag{4.3}$$

with $[\cdot]_a^b$ being defined by $\max\{a, \min\{b, \cdot\}\}$.

Setting α to 3.5 and β to 0.3, Figure 4.1 shows some samples of our packet reception ratio model, for parameters $\delta = 20$, $D_1 = 10$ and $D_2 = 30$ (we will use the same setting for our simulations). The connected, transitional, and disconnected regions can easily be distinguished by D_1 and D_2, while the transitional region shows some variation, as intended. Although those parameters were chosen arbitrarily, they seem quite reasonable with respect to the simulation area that we will consider later. But even other parameters have led to similar simulation results.

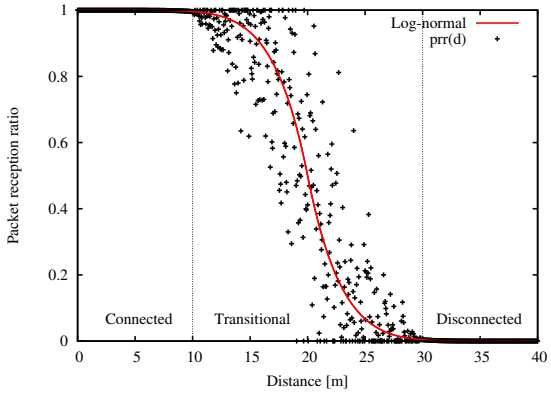


Figure 4.1: Samples of the PRR model ($\alpha = 3.5$, $\beta = 0.3$, $D_1 = 10$, $D_2 = 30$)

4.3.2 Link Asymmetry

Link asymmetry is modeled by evaluating $prr(d)$ twice: separately for each direction. Thus, the packet reception ratio on a link between a node i and a node j is a 2-tuple $[prr_{i,j}(d), prr_{j,i}(d)]$, assuming both nodes are d meters apart from each other. Note that due to the Gaussian variable both values may be different if d is within the transitional region. For the parameter set used above, the average difference in the packet reception ratio is about 0.13 if only the transitional region is considered. The probability that the difference per link will be lower than a given threshold is shown in Figure 4.2. For example, at about 25%, the difference between the packet reception ratios on an asymmetric link is more than 0.2.

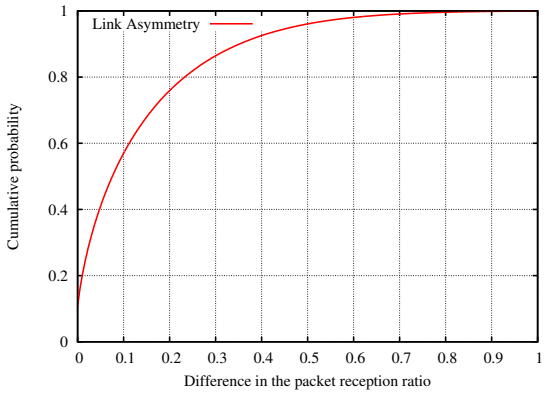


Figure 4.2: Cumulative probability of link asymmetry

4.3.3 Energy Model

To ascertain the amount of energy consumed by a radio transceiver, we apply the following energy model. For each packet transmitted by a sending node to one or more receivers in its neighborhood, the energy is calculated as

$$e = e_{tx} + n \cdot e_{rx} + (N - n) \cdot e_{rx}^{h}, \tag{4.4}$$

where e_{tx} and e_{rx} denote the amount of energy required to send and receive, n the number of addressed nodes which should receive the packet, and N the total number of neighbors in the transmission range. e_{rx}^{h} quantifies the amount of energy required to decode only the packet header. By using appropriate MAC protocols like S-MAC [270] or WiseMAC [81], nodes whose addresses are not contained in the packet header are able to turn their radio units off during the ongoing transmission of the packet. However, it is still necessary to receive the complete header in order to get information about addressed nodes and the packet length. If such a MAC protocol is not used, e_{rx}^{h} would be equal to e_{rx}. In this case, the energy consumption of idle listening must also be considered.

According to the *first order model* described in [114], e_{tx} and e_{rx} are defined as

$$e_{tx}(d, k) = (e_{elec} + e_{amp} \cdot d^{\gamma}) \cdot 8k \tag{4.5}$$

$$e_{rx}(k) = e_{elec} \cdot 8k \tag{4.6}$$

for a distance d and a k-byte message. γ denotes the path loss exponent that we set to 2. We also set $e_{elec} = 50$ nJ/bit as the energy needed to run the radio transceiver's circuity, and $e_{amp} = 100$ pJ/bit/m^2 as that consumed by the transmit amplifier. Since it is assumed that the transmission power of all sensor nodes is fixed, d is set to the maximum transmission range of 30 m. For example, a packet with 32 bytes would require a reception energy of $e_{rx} = 12.8$ μJ and a transmitting energy of $e_{tx} = 35.84$ μJ. Assuming a header size of 8 bytes, e_{rx}^{h} would be 1.6 μJ.

4.3.4 Assumptions

In the following analyses and simulations we consider a stationary WSN of size 200 $\times$ 200 m^2 with a maximum transmission range of 30 m. Nodes are scattered according to a uniform distribution. It is assumed that each node in the network will generate a data report that will then be forwarded to one predefined sink node. The message length is assumed to be 32 bytes, including an 8-byte packet header, which we consider to be a proper size for data packets in a WSN.

The energy used to receive and transmit data is modeled according to the energy model presented above. Other sources of energy consumption like sensing, processing, and idle listening are neglected as they are quite similar for each node in the network and nearly independent of the forwarding strategy. Thus, nodes not participating in any communication are assumed to turn their radios off. Other MAC-layer behavior such as contention, duty cycles, or packet buffering are not addressed as we believe they are orthogonal problems. This seems to be very reasonable for most WSNs, which show relatively light traffic and contention.

Furthermore, we assume that a node knows about the packet reception ratios on wireless links to its neighbors, e.,g., through packet reception measurements performed earlier. Link estimators, as analyzed in [244, 258], could be used to provide this information. For example, Woo and Culler propose a window-based link estimator that computes the packet success rate over a predefined time period by using an exponentially-weighted moving average.

It is also assumed that each node periodically broadcasts *beacon* messages in order to inform neighbors that it is still alive. Besides the node's id and a sequence number, beacons contain information concerning the packet reception ratios the node has measured in order to identify asymmetric links. In addition, information about forwarding paths and corresponding forwarding metrics are included, too. In so doing, the periodic beacon process is used to establish a forwarding tree in the network. As the costs required to periodically send beacons are the same for each forwarding strategy, they will be ignored in the following.

4.3.5 Metrics

We evaluate all forwarding strategies with two metrics, namely the *end-to-end delivery ratio* and the *energy efficiency*. Assuming that the radio transmission range of each node is r, we denote the number of nodes per πr^2 area by the node density μ. The delivery ratio denoted by E_i^r quantifies the fraction of packets originating at an arbitrary node i that are properly received at the sink. The corresponding energy consumed by receiving and transmitting packets along the forwarding path is denoted by E_i^e.

The ratio between the number of delivered bits and the consumed energy on the forwarding path then defines the energy efficiency E_i^{eff}, which is calculated as

$$E_i^{eff} = \frac{E_i^r \cdot 8k}{E_i^e} \tag{4.7}$$

for a single-packet transmission with a packet length of k bytes.

4.4 Analysis of Hop- and PRR-Based Forwarding Strategies

In the following, we start by analyzing two very simple forwarding strategies which are based on the path length, defined as the number of forwarding nodes (hops), and on the packet reception ratio to the first forwarding node. By blacklisting bad neighbors, we investigate about how much the end-to-end delivery ratio and the energy efficiency can be improved.

4.4.1 Hop-Based Forwarding

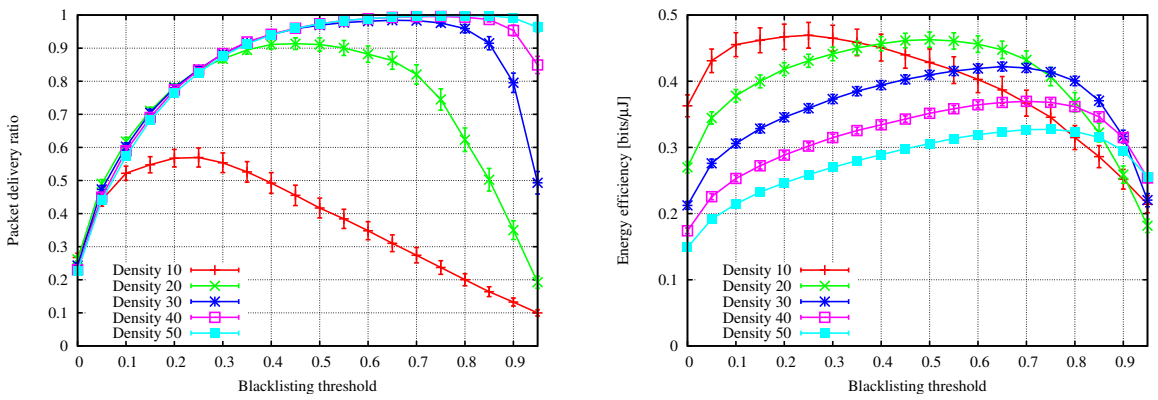
In order to find a path from several source nodes to a fixed destination, a widely used method is to establish the reverse path by using hop counters: The destination node floods the network with a special routing message containing the number of hops the node is away. Each node receiving a

message with a lower hop counter updates its routing table and stores the new forwarding node and hop counter. Then, the node broadcasts its updated information to its neighborhood. Once no further changes occur, the forwarding path is defined by following the path with decreasing hop counters.

However, especially in the field of sensor networks where low-power radios are used, nodes might have many neighbors with lossy links. Traditional hop-based forwarding algorithms neglect link qualities and simply select the node with the minimum hop counter as the forwarder. But as it is likely that such nodes will be far away, many retransmissions may later be necessary, causing a high energy consumption.

One possibility to overcome this problem is to blacklist bad nodes, thereby preventing them from becoming forwarders [161, 258]. From among all neighbors of a node, i. e., nodes with a packet reception ratio greater or equal to the blacklisting threshold, the node with the minimum hop counter will be selected as the forwarder. In case two or more neighbors have the same hop counter, the one with the higher reception ratio is selected. However, finding the best blacklisting threshold may be difficult. Several environmental conditions like the node density and link qualities may influence the performance.

Figure 4.3 illustrates the impact of the node density for different blacklisting thresholds on the delivery ratio and energy efficiency, averaged over all nodes randomly placed in a 200 $\times$ 200 m^2 area. We performed 500 simulation runs using the packet reception model described in Section 4.3.1. The number of retransmissions in case of packet loss, which we denote by R, was set to three.



(a) Packet delivery ratio

(b) Energy efficiency

Figure 4.3: Hop-based forwarding using different blacklisting thresholds ($R = 3$)

In Figure 4.3(a), the end-to-end packet delivery ratio averaged over all nodes in the network is shown, together with the 0.95 quantile. Without blacklisting nodes, the delivery ratio is quite low, as many nodes use forwarders which have poor links but a small hop counter. By blacklisting such nodes, the delivery ratio increases up to the point when blacklisting is no longer favorable because of occurring disconnections. Especially for low node densities where each node only has a few neighbors, blacklisting causes rapid disconnection, decreasing the delivery ratio substantially.

In the same way, the energy efficiency is influenced as shown in Figure 4.3(b). However, a blacklisting threshold that maximizes the packet delivery ratio need not implicitly yield the highest energy

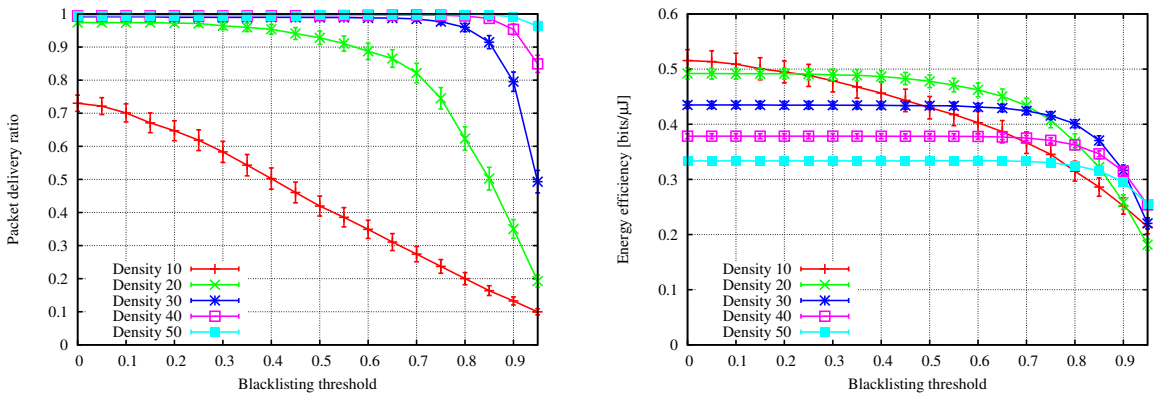
efficiency. A low blacklisting threshold leads to low packet reception ratios and thus causes a high energy consumption due to packet retransmissions. Thus, the energy efficiency defined by the ratio of both is low, too. By increasing the threshold, forwarding paths become longer, but at the same time the quality of the forwarding links improves. Although in this case the energy consumption initially increases because more forwarding links are used, the costs of additional nodes on a forwarding path are compensated eventually due to fewer retransmissions.

As both figures show, blacklisting in conjunction with hop-based forwarding clearly improves the end-to-end packet delivery ratio as well as the energy efficiency and is thus very useful. However, the optimal blacklisting threshold depends on the node density, which makes it quite difficult to specify it beforehand.

4.4.2 PRR-Based Forwarding

In PRR-based forwarding, a node i selects a forwarder j according to its distance (the hop counter), and in addition, the packet reception ratio to the forwarding node. By minimizing $\frac{hops}{prr_{i,j}prr_{j,i}}$, the best forwarding node is selected. The idea behind this metric is to downgrade neighbors with low hop counters or poor links, which should minimize the influence of blacklisting on the forwarding performance. An advantage is that nodes having poor links will still be considered and not blocked completely. Thus, disconnections caused by blacklisting too many nodes may be prevented. Nevertheless, blacklisting could still help to improve the delivery ratio and energy efficiency since very bad links might be avoided, independent of assigned hop counters.

The effects of neighbor blacklisting on the end-to-end packet delivery ratio and the resulting energy efficiency are shown in Figure 4.4. For low node densities, blacklisting does not improve the packet delivery ratio at all but only causes disconnections. For higher densities, the delivery ratio is improved slightly, but the improvement is marginal and can actually be neglected. Thus, the resulting energy efficiency rather decreases if blacklisting is performed, either due to disconnections or to longer forwarding paths.



(a) Packet delivery ratio

(b) Energy efficiency

Figure 4.4: PRR-based forwarding using different blacklisting thresholds ($R = 3$)

A comparison of hop-based forwarding and PRR-based forwarding shows that both strategies nearly achieve the same optima for high node densities, with PRR-based forwarding performing slightly better. However, the fact that it is basically independent from blacklisting turns it into a very practical alternative since it is not necessary to know the right blacklisting threshold *a priori*. Furthermore, since it does not strictly block bad nodes, disconnections are not caused accidentally.

4.5 Energy-Efficient Forwarding

Energy-efficient forwarding aims to find the most energy-efficient path in the network that trades off the end-to-end delivery ratio and the energy cost. By examining each of a node i's neighbors j, the node that maximizes E_i^{eff} is selected as the forwarder. In contrast to the previously proposed forwarding strategies, we now take the end-to-end reception rate and energy consumption into account simultaneously.

4.5.1 Single-Link Energy-Efficient Forwarding

As in the hop-based and PRR-based forwarding strategies, the network sink first initializes the establishment of reverse paths by broadcasting a special routing message, which is "flooded" throughout the network by using the nodes' beacons sent periodically. This implies that each beacon will contain information regarding the current end-to-end delivery ratio and the forwarding cost of the node sending the beacon. By using this information received form adjacent neighbors, a node i is thus able to calculate E_i^r, E_i^e, and E_i^{eff} for the best forwarding path a neighbor j provides. The neighbor j that maximizes E_i^{eff} is then stored in the routing table of node i as the forwarder, together with information about E_i^r and E_i^e. If the energy efficiency could be improved or E_i^r and E_i^e have changed with respect to the current forwarder, this will be announced to the node's neighbors with the next beacon.

In contrast to the forwarding strategy we consider in the next section, we call this strategy *single-link energy-efficient forwarding* (SEEF). *Single-link* refers to the case in which a message is forwarded over a single link to one forwarding node. On the other hand, the following *multi-link* forwarding strategy will exploit the broadcast characteristics of the wireless medium by using multiple forwarding links.

4.5.2 Multi-Link Energy-Efficient Forwarding

In the *multi-link energy-efficient forwarding* (MEEF) strategy, packets may be sent to more than one forwarding node. The idea there is to exploit the broadcast characteristics the wireless communication channel provides. In general, nodes that are not addressed in the header of a packet as the destination temporarily turn their radios off to save energy. However, it might be more efficient if some nodes were to stay awake and overhear the packet transmission completely.

In case some nodes are not able to receive the packet correctly, retransmitting the entire packet from the source will not be necessary if there is another node on the way that has received it successfully. In order to prevent the packet's being forwarded by multiple nodes, we use a simple polling approach

that is employed by the source. At first, all potential forwarders are ordered according to their energy efficiency. If the first forwarding node does not acknowledge the packet, we assume the packet got lost. By polling the next node on the list, the source informs the second forwarder that it should forward the packet instead, and so on. Only if none of the nodes answers, will the packet be retransmitted by the source.

Although this approach usually prevents multi-path forwarding, it is still possible that a packet will be forwarded by two or even more nodes at the same time: Due to link asymmetries, it may happen that a packet will have been correctly received by the first forwarder but the sender did not get an acknowledgement. It thus will poll the second forwarding node, which in turn will start forwarding the packet.

Even though such an "unintentional" multi-path forwarding might be robust against different kinds of network failures, it might still be inefficient concerning its energy consumption [79, 90, 166]. However, the described polling approach reduces the impact of multi-path forwarding substantially, as packets are forwarded by more than one node only if acknowledgments get lost. Nevertheless, the additional energy costs for this case, as well as the energy cost for polling nodes need to be taken into account. Otherwise, a node will be unable to decide reliably whether or not the energy efficiency of its forwarding path can be improved by using additional forwarders.

Concerning the addressing used by multi-link forwarding, the appropriate node creates an ordered set of potential forwarders, the *forwarder list*, which is added to the packet header. The details of how to determine the ordered set will be described later. Nodes contained in the forwarder list stay awake during the complete transmission and try to receive the entire packet. All other nodes are assumed to temporally turn their communication radios off, like in SEEF. As each additional forwarding node must receive the packet in order to work in a so-called *backup* mode, more energy is consumed. But at the same time, additional nodes may improve the delivery ratio and also the overall energy efficiency. Thus, whether more than one forwarder should be selected mainly depends on the packet reception ratios on the forwarding links, as well as on the end-to-end delivery ratios and energy costs of the selected forwarders.

In the following, we present the mathematical details of single-link and multi-link forwarding and analyze how the end-to-end delivery ratio, the energy cost, and the energy efficiency can be calculated. We consider the case of infinite as well as of finite retransmissions.

4.5.3 Analysis of the Infinite Retransmissions Case

We start by analyzing the case of infinite retransmissions, i. e., a node i will repeatedly send a data packet to its forwarding node (or to the forwarder set in case of MEEF) until an acknowledgement is received. The acknowledgement can either be sent explicitly or implicitly by piggy-backing it with the next forwarder's data packet [191]. However, in the following, we consider only the case of explicit acknowledgements.

As the number of retransmissions is infinite, each packet will finally reach its destination, leading to a delivery ratio of one. Thus, maximizing E_i^{eff} is equal to minimizing the energy consumption E_i^e.

Single-Link Energy-Efficient Forwarding

In the case of single-link forwarding, the calculation of the energy consumption is based on the probability tree shown in Figure 4.5. F and $\overline{F}$ denote the events of "forwarded packet received correctly, respectively not correctly". The events of "acknowledgement received correctly, respectively not correctly" are denoted by A and $\overline{A}$. In addition, let E_j^e be the energy required by node j to forward a packet, and $prr_{i,j}$ and $prr_{j,i}$ be the packet reception ratios on the link between node i and j, and vice versa.

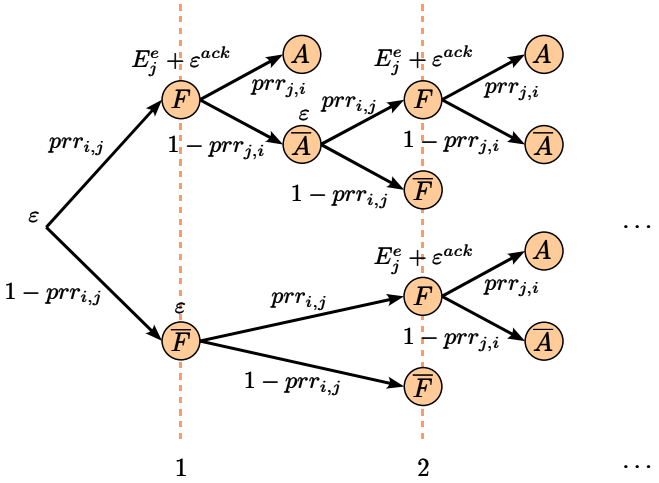


Figure 4.5: Probability tree for the energy consumption of SEEF

According to the energy model presented in Section 4.3.3, the energy consumed by transmitting single data packets and acknowledgements is expressed as

$$\begin{aligned} \varepsilon &= e_{tx}(d,k) + e_{rx}(k) + (N-1)e_{rx}(8) \quad \text{and} \\ \varepsilon^{ack} &= e_{tx}(d,8) + e_{rx}(8), \end{aligned}$$

for a data packet size of k bytes and a maximum transmission range of d, which is set to 30 m, as defined by the packet reception model from Section 4.3.1. The length of acknowledgements, as well as that of a packet header, is assumed to be 8 bytes. To simplify matters, we assume that each packet received by a node will always be forwarded, even if it was retransmitted and maybe forwarded before. Thus, we do not model any packet history buffers in which the last packets are stored.

According to Figure 4.5, the energy efficiency can be calculated as follows: While the end-to-end delivery ratio is one (due to infinite retransmissions), the energy required for single-link forwarding is

$$\begin{aligned} E_i^e &= \varepsilon + prr_{i,j}\left(E_j^e + \varepsilon^{ack} + (1 - prr_{j,i})E_i^e\right) + (1 - prr_{i,j})E_i^e \\ &= \varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack}) + (1 - prr_{i,j}prr_{j,i})E_i^e \\ &= \frac{\varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack})}{prr_{i,j}prr_{j,i}}. \end{aligned} \tag{4.8}$$

Thus, the energy efficiency defined by Equation 4.7 is $8k/E_i^e$.

Multi-Link Energy-Efficient Forwarding

Extending the calculation to the case of multi-link forwarding, where an ordered set Ω_i of n potential forwarders is used, leads to a corresponding probability tree illustrated in Figure 4.6. $\alpha(j)$ denotes the position of node j in Ω_i and $\hat{\alpha}(k)$ denotes the forwarder at position k.

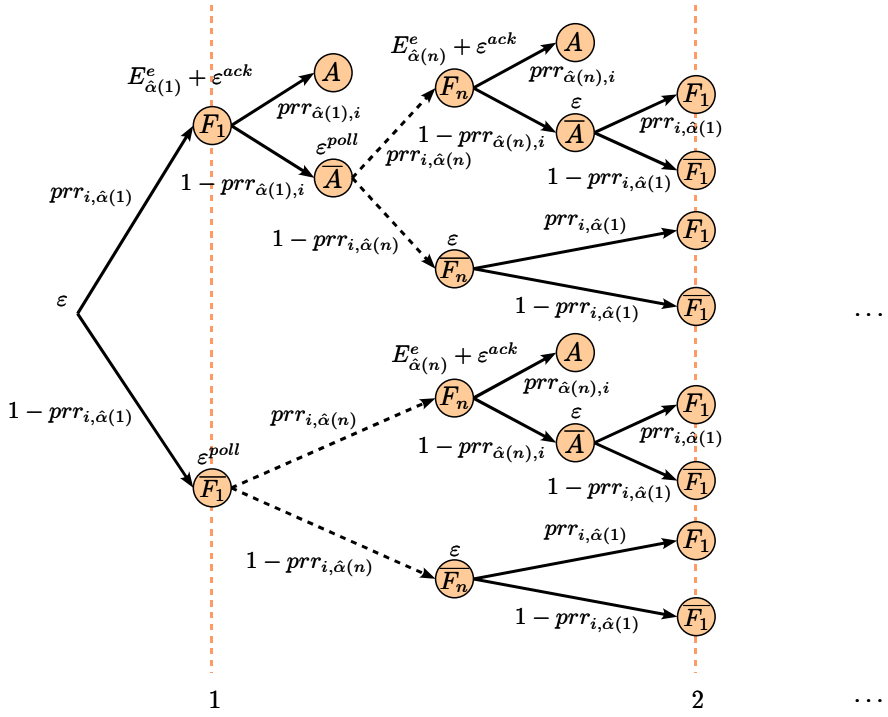


Figure 4.6: Probability tree for the energy consumption of MEEF

Similarly, the energy consumed by transmitting data packets, acknowledgements, and polling messages is defined as

$$\begin{aligned} \varepsilon &= e_{tx}(d,k) + ne_{rx}(k) + (N-n)e_{rx}(8), \\ \varepsilon^{ack} &= e_{tx}(d,8) + ne_{rx}(8), \quad \text{and} \\ \varepsilon^{poll} &= e_{tx}(d,8) + ne_{rx}(8). \end{aligned}$$

We assume that acknowledgements and polling messages are received only by *active* nodes, i. e., nodes that did not turn their communication radio off during the forwarding of data packets. In addition, we assume for the sake of simplicity that on average, the number of nodes that will be affected by acknowledgements will be equal to the size of the forwarder set of node i[2].

The energy consumption for an arbitrary forwarding set Ω_i is then calculated as

$$E^e_i = \varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j}(E^e_j + \varepsilon^{ack}) \right) + \rho_{i,n} E^e_i$$

[2]Note that the actual cost for receiving acknowledgements is determined by the number of active neighbors the sender of an acknowledgement has.

$$= \frac{\varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j}(E_j^e + \varepsilon^{ack}) \right)}{1 - \rho_{i,n}}, \tag{4.9}$$

where $\rho_{i,k}$ is defined as

$$\rho_{i,k} = \begin{cases} 0 & k = 0, \\ \prod_{\forall j \in \Omega_i, \alpha(j) \leq k} (1 - prr_{i,j} prr_{j,i}) & k > 0, \end{cases}$$

and $|x|$ denotes a Boolean operator returning one if x is true, and zero otherwise. As in the single-link case, the energy efficiency is calculated from $8k/E_i^e$.

4.5.4 Analysis of the Finite Retransmissions Case

In the case of finite retransmissions, each node only tries $R+1$ times to send a packet to its forwarding nodes, with R being the number of retransmissions. If the packet cannot be delivered by then, it will be discarded. Compared to the previous section, this case seems to be more realistic, as in practice a node would not be able to send a packet an infinite number of times. For example, if the communication between a node and its forwarder is disturbed for a longer period, a node will consume all its energy by trying to forward a packet, which does not make sense. Thus, the assumption of a 100% delivery ratio is not really realistic. Moreover, if all nodes never stop retransmitting undelivered packets, the influence of network congestion can no longer be ignored.

Single-Link Energy-Efficient Forwarding

In the single-link case, the end-to-end delivery ratio of node i can be calculated as follows: Let $\hat{E}_r^k$ be the delivery ratio if up to k transmissions are used. According to Figure 4.5, we can express $\hat{E}_r^1$ to $\hat{E}_r^{R+1}$ iteratively as

$$\hat{E}_r^{R+1} = prr_{i,j} E_j^r + (1 - prr_{i,j}) \hat{E}_r^R \tag{4.10}$$

$$\hat{E}_r^R = prr_{i,j} E_j^r + (1 - prr_{i,j}) \hat{E}_r^{R-1} \tag{4.11}$$

$$\vdots \tag{4.12}$$

$$\hat{E}_r^1 = prr_{i,j} E_j^r. \tag{4.13}$$

Note that in terms of the delivery ratio, the reverse link used for acknowledgements is not important. Because even if an acknowledgement does get lost and a packet is successfully forwarded twice, the second packet will be redundant and will not improve the delivery ratio. Compared to the calculation in Equation 4.8, we can therefore neglect $prr_{j,i}$.

With $E_i^r = \hat{E}_r^{R+1}$, the end-to-end delivery ratio is then

$$E_i^r = prr_{i,j} E_j^r \sum_{k=0}^{R} (1 - prr_{i,j})^k$$

$$= E_j^r \left(1 - (1 - prr_{i,j})^{R+1}\right). \tag{4.14}$$

Similar to Section 4.5.3, the energy consumption is calculated. $\hat{E}_e^k$ denotes the energy required by node i to forward a packet towards the sink if up to k transmissions are used. Then, $\hat{E}_e^{R+1}$ is iteratively computed as

$$\begin{aligned}
\hat{E}_e^{R+1} &= \varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack}) + (1 - prr_{i,j}prr_{j,i})\hat{E}_e^R \\
\hat{E}_e^R &= \varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack}) + (1 - prr_{i,j}prr_{j,i})\hat{E}_e^{R-1} \\
&\vdots \\
\hat{E}_e^1 &= \varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack}).
\end{aligned} \tag{4.15}$$

Thus, the end-to-end energy consumption is

$$\begin{aligned}
E_i^e &= \left(\varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack})\right) \sum_{k=0}^{R} (1 - prr_{i,j}prr_{j,i})^k \\
&= \frac{\left(\varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack})\right)\left(1 - (1 - prr_{i,j}prr_{j,i})^{R+1}\right)}{prr_{i,j}prr_{j,i}}.
\end{aligned} \tag{4.16}$$

According to Equation 4.7, the energy efficiency defined as $8k \cdot E_i^r / E_i^e$ is

$$E_i^{eff} = \frac{8k \cdot E_j^r \left(1 - (1 - prr_{i,j})^{R+1}\right) prr_{i,j}prr_{j,i}}{\left(\varepsilon + prr_{i,j}(E_j^e + \varepsilon^{ack})\right)\left(1 - (1 - prr_{i,j}prr_{j,i})^{R+1}\right)}. \tag{4.17}$$

Multi-Link Energy-Efficient Forwarding

Analogous to the analyses done so far, we can extend the calculations to the multi-link case using R retransmissions. According to Figure 4.6, the end-to-end delivery ratio is computed as

$$\begin{aligned}
\hat{E}_r^{R+1} &= \sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r + \overline{\rho}_{i,n} \hat{E}_r^R \\
\hat{E}_r^R &= \sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r + \overline{\rho}_{i,n} \hat{E}_r^{R-1} \\
&\vdots \\
\hat{E}_r^1 &= \sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r,
\end{aligned} \tag{4.18}$$

with

$$\overline{\rho}_{i,k} = \begin{cases} 0 & k = 0, \\ \prod_{\forall j \in \Omega_i, \alpha(j) \leq k} (1 - prr_{i,j}) & k > 0. \end{cases}$$

Simplifying these Equations leads to

$$\begin{aligned} E_i^r &= \left(\sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r \right) \sum_{k=0}^{R} (1-\overline{\rho}_{i,n})^k \\ &= \frac{\left(\sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r \right) \left(1-\overline{\rho}_{i,n}^{R+1} \right)}{1-\overline{\rho}_{i,n}}. \end{aligned} \tag{4.19}$$

In the same way, the energy consumption for the multi-link case is calculated as

$$\begin{aligned} \hat{E}_e^{R+1} &= \varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j} (E_j^e + \varepsilon^{ack}) \right) + \rho_{i,n} \hat{E}_e^R \\ \hat{E}_e^R &= \varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j} (E_j^e + \varepsilon^{ack}) \right) + \rho_{i,n} \hat{E}_e^{R-1} \\ &\vdots \\ \hat{E}_e^1 &= \varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j} (E_j^e + \varepsilon^{ack}) \right) \end{aligned} \tag{4.20}$$

that leads to

$$E_i^e = \frac{\left(\varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j} (E_j^e + \varepsilon^{ack}) \right) \right) \left(1-\rho_{i,n}^{R+1} \right)}{1-\rho_{i,n}}. \tag{4.21}$$

Finally, the energy efficiency is expressed as

$$E_i^{eff} = \frac{8k \cdot \left(\sum_{\forall j \in \Omega_i} \overline{\rho}_{i,\alpha(j)-1} prr_{i,j} E_j^r \right) \left(1-\overline{\rho}_{i,n}^{R+1} \right) \left(1-\rho_{i,n} \right) / \left(1-\overline{\rho}_{i,n} \right)}{\left(\varepsilon + \sum_{\forall j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1| \varepsilon^{poll} + prr_{i,j} (E_j^e + \varepsilon^{ack}) \right) \right) \left(1-\rho_{i,n}^{R+1} \right)}. \tag{4.22}$$

4.5.5 Analysis of the Optimal Number of Forwarders for MEEF

Taking Equations 4.17 and 4.22 into account, we can now analyze how the link quality influences the number of selected forwarders and when it would be most efficient to select more than one forwarding node. Let us consider a simple example consisting of a 1-hop neighborhood with N nodes, where N is set to 20. We assume that all nodes will have the same packet reception ratio p and require the same amount of energy to forward packets, i. e., without loss of generality, we set $E_j^e = 0$. Furthermore, all links are symmetric.

The energy efficiency in Equation 4.22 then changes to

$$E_i^{eff} = \frac{8k \cdot \left(1-(1-p)^{n(R+1)} \right) \left(1-(1-p^2)^n \right)}{\left(\varepsilon + \varepsilon^{ack} \left(\frac{1-(1-p^2)^n}{p} \right) + \varepsilon^{poll} \left(\frac{1-(1-p^2)^n}{p^2} - 1 \right) \right) \left(1-(1-p^2)^{n(R+1)} \right)}, \tag{4.23}$$

which is plotted in Figure 4.7 for a different number of forwarding nodes and packet reception ratios, using three retransmissions and a packet size of $k = 32$ bytes.

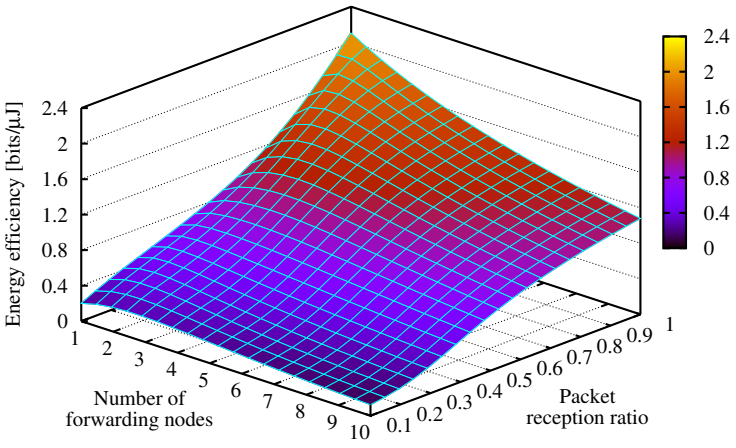
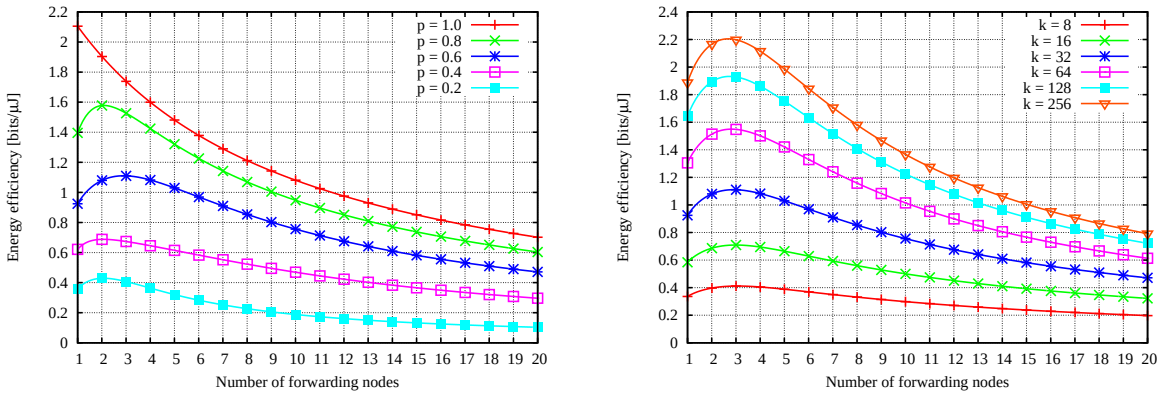


Figure 4.7: Energy efficiency for a different number of forwarders and packet reception ratios ($R = 3$, $k = 32$)

As Figure 4.7 illustrates, with decreasing reception ratios the energy efficiency tends to decrease, too. In this case, the packet delivery ratio worsens, while at the same time more energy is spent on retransmissions. The same is true for the number of forwarding nodes, as more forwarders will require more energy in order to receive data packets. However, there is a trade-off since using more forwarding nodes will increase the delivery ratio, even if more energy is needed. Thus, in the following we would like to find the optimal number of forwarders needed to maximize the energy efficiency.

Figure 4.8 provides another view of Figure 4.7 for different packet reception ratios. Since for a perfect packet reception ratio of $p = 1$ additional forwarding nodes only require more energy, the energy efficiency decreases monotonically for $n \geq 1$, as shown in Figure 4.8(a). However, for $p \leq 0.8$, using more than one forwarder is more efficient. Even if other packet sizes are considered, the energy efficiency can be improved by multi-link forwarding. As illustrated in Figure 4.8(b), larger packets also increase the energy efficiency as the ratio between the entire packet size and the packet header grows, i. e., the overhead caused by the packet header slightly decreases, which in turn increases the energy efficiency.

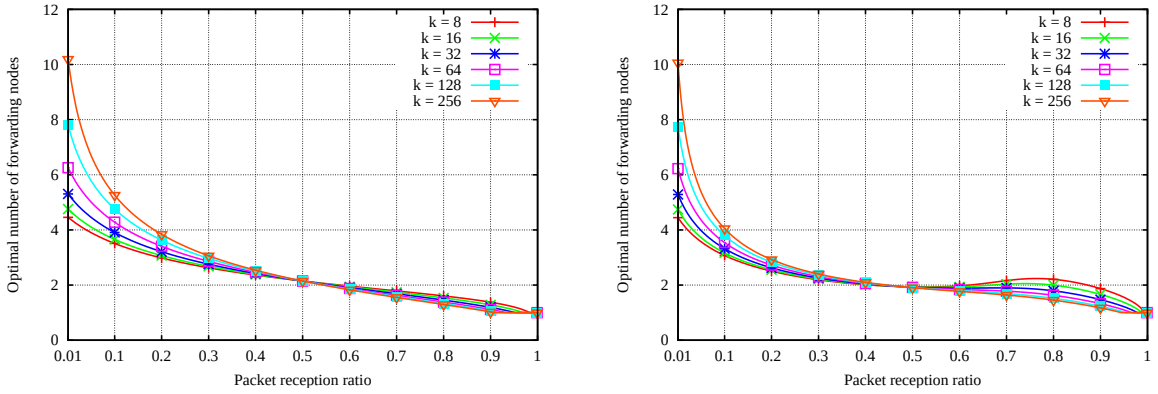


(a) $k = 32$ (b) $p = 0.6$

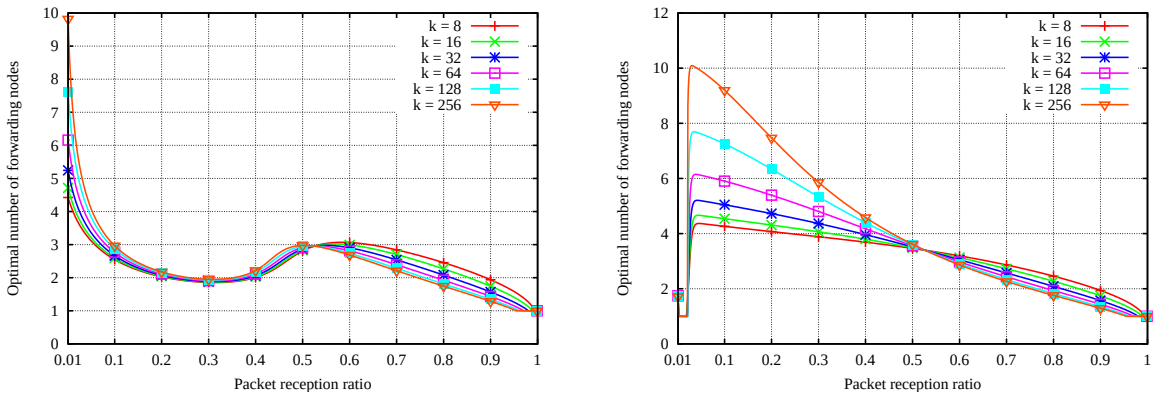
Figure 4.8: Energy efficiency for different packet reception ratios and packet sizes ($R = 3$)

Calculating the optimal number of forwarding nodes, which is denoted by n^*, can be done by solving $\frac{\partial E_i^{eff}}{\partial n} = 0$. Figure 4.9 shows how n^* is influenced by the packet reception ratio and the number of retransmissions. If retransmissions are not used (see Figure 4.9(a)), the optimal number of forwarders will monotonically increase for decreasing packet reception ratios, while using more than one forwarder will become beneficial if $p \leq 0.5$. If the number of retransmissions is increased (Figure 4.9(b) to Figure 4.9(d)), we can observe an interesting effect. At first, n^* drops if the packet reception ratio increases, as higher delivery ratios require fewer forwarding nodes in order to maximize the energy efficiency. However, at some point (which depends on the number of allowed retransmissions) the maximum efficiency is achieved again for higher n^*. In this range, using retransmissions and more forwarders will increase the delivery ratio more than it will cause additional energy costs. Thus, the energy efficiency reaches its maximum for higher n^*. If, however, the packet reception ratio increases further, neither retransmissions nor additional forwarders will compensate the energy cost any longer. Hence, the optimal number of forwarders n^* will again decrease until it reaches 1 for $p = 1$.

Figure 4.9 also shows the impact of different packet sizes on n^*. For high packet reception ratios, using additional forwarding nodes fails to improve the delivery ratio as it does for low reception ratios. Small packet sizes, which cause less energy consumption, thus allow for higher n, improving the delivery ratio as much as the additional energy costs are compensated. However, the additional cost caused by larger packets can only be compensated if fewer forwarding nodes are used.



(a) $R = 0$ (b) $R = 1$



(c) $R = 3$ (d) $R = \infty$

Figure 4.9: Optimal number of forwarders for different packet reception ratios and packet sizes

In contrast, in the case of low packet reception ratios, a higher energy consumption can be compensated if n increases, as the delivery ratio grows more than for high reception ratios. Furthermore, the overhead caused by acknowledgements and polling messages will decrease in relation to the data packet size if larger packets are used[3]. As the impact of these control packets will grow if lower packet reception ratios are considered (which is mainly determined by the energy required for polling), more forwarding nodes can be used if the size of data packets is larger. In this case, the *relative* amount of energy spent on control packets with respect to data packets is lower. The relative improvement in the delivery ratio thus compensates a larger energy increase, as it does for smaller data packets.

In conclusion, using additional forwarders in MEEF may indeed increase the energy efficiency if the packet reception ratios on forwarding links are not perfect. Additional energy costs for multi-path forwarding and polling may be compensated by an increase in the end-to-end delivery ratio. Moreover, if the maximum number of retransmissions is limited to three ($R = 3$), using more than three forwarding nodes will not further improve the energy efficiency. This is quite useful for practical implementations, as only forwarder sets with up to three nodes need to be analyzed.

4.6 Simulations

With the mathematical analyses on hand, we now investigate by means of simulations how well the different forwarding schemes work. We have implemented the following strategies:

- **MEEF** As described in Section 4.5, MEEF attempts to maximize the end-to-end energy efficiency E_i^{eff} for a node i according to Equations 4.9 and 4.22, respectively. Each node gathers appropriate data from its neighborhood. It then evaluates E_i^{eff} for different forwarder sets of size n and selects the nodes that lead to a maximum value. Note that the forwarder set defines an order among the nodes, according to which a packet is forwarded. If the first node of the forwarding set does not acknowledge the packet, the next node will be polled until no forwarding node remains, and the packet is retransmitted[4].

- **SEEF** Except for the fact that just one forwarding node and not a set of potential forwarders is considered, SEEF works similar to MEEF. Thus, no energy is consumed by multi-path forwarding or polling.

- **MT Forwarding** attempts to minimize the overall packet transmissions along a source-to-sink path. As proposed in [64] and [258], MT forwarding evaluates $\frac{1}{prr_{i,j}prr_{j,i}}$ for each neighbor j of a node i and selects as its forwarder the neighbor that minimizes this expression. As the metric assumes *infinite* retransmissions, we extend the calculation by considering the actual number of transmissions, which leads to $\frac{1-(1-prr_{i,j}prr_{j,i})^{R+1}}{prr_{i,j}prr_{j,i}}$. We call this extended strategy MT2 Forwarding.

[3] Note that both acknowledgements and polling messages have a fixed size of 8 bytes.

[4] Note that the polling mechanism is only required if the first node does not receive the packet correctly.

- **PRR-based Forwarding** tries to minimize $\frac{hops}{prr_{i,j}prr_{j,i}}$, as a trade-off between link quality and distance. Since blacklisting has actually no improving impact on the end-to-end delivery ratio and the energy efficiency, it is not applied.

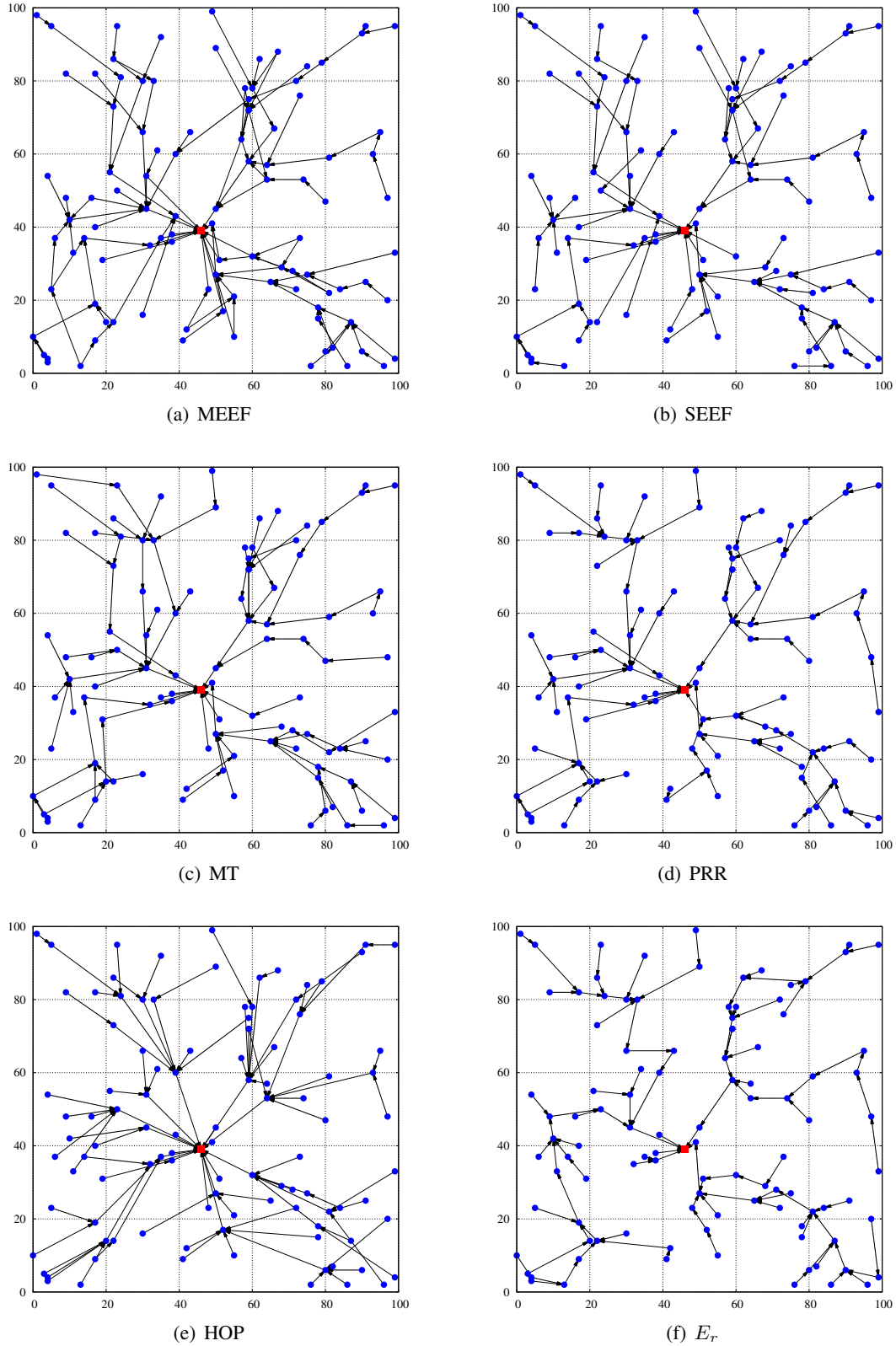
- **E_r-based Forwarding** focuses on the end-to-end delivery ratio only [97] and attempts to maximize E_i^r as it is calculated in Equation 4.14. E_r-based forwarding is thus expected to achieve the best delivery ratios among all single-link forwarding strategies. However, the energy consumption might be high if only short-distance links are used.

- **Hop-based Forwarding** is the simplest manner of forwarding, which considers the hop counters of adjacent neighbors, indicating how far away the sink is. The neighbor with the lowest hop counter then becomes the forwarder of a node i. If the hop counters of two neighbors are equal, the node with the better packet reception ratio will be selected. Concerning the forwarding path length, hop-based forwarding should thus achieve the best results.

- **Hop*-based Forwarding** works like hop-based forwarding but additionally performs a blacklisting of bad neighbors. The blacklisting threshold is optimized according to the simulation results obtained from Section 4.4 such that the end-to-end energy efficiency is optimized. Thus, it gives an upper bound for hop-based forwarding without blacklisting.

Figure 4.10 illustrates these forwarding strategies in a 100 $\times$ 100 m^2 sample network for a density of 30 nodes and one sink node, which is indicated by a red circle. All established forwarding links are depicted as arrows and give a first impression of how the different forwarding paths may look.

In Figure 4.10(a), the forwarding graph of MEEF is shown. It can be seen that several nodes establish paths by exploiting multi-link forwarding, particularly in dense neighborhoods. However, consistent with the results from the last section, all nodes use at most two forwarders. Compared to the SEEF tree that is depicted in Figure 4.10(b), we can see that most of the SEEF paths are congruent with MEEF. That is mainly due to the fact that the energy efficiency of SEEF is always a lower bound for MEEF, which uses additional forwarders only if they improve the efficiency of a forwarding path. The trees of MT and PRR-based forwarding are illustrated in Figure 4.10(c) and Figure 4.10(d). In some areas they are quite different, while other areas are similar to the forwarding paths used in SEEF and MEEF. Especially where the node density is sparse, the same forwarding nodes are selected because finding sufficient alternatives to well-connected neighbors is unlikely for low densities. Concerning the path length, PRR-based forwarding seems to prefer longer paths, although its forwarding metric incorporates the hop counter of a forwarding node. However, as it considers the packet reception ratio only on the first link to a forwarder and not the end-to-end delivery ratio, selecting well-connected nodes becomes preferable, even if this increases the hop counter. On the other hand, hop-based forwarding looks quite different (see Figure 4.10(e)). We can clearly see that its forwarding paths are shorter. Even far-away nodes reach the sink in at most three hops because several links are long-distance ones. At last, Figure 4.10(f) shows the forwarding tree of E_r-based forwarding. Since this strategy only considers the end-to-end delivery ratio of a forwarding path independently of the path length, short-distance links are used in the majority of cases. While such links are likely to show better packet reception ratios, the total number of packet transmissions may be very high, even if fewer retransmissions per link may be necessary.



(a) MEEF (b) SEEF

(c) MT (d) PRR

(e) HOP (f) E_r

Figure 4.10: Sample network showing different forwarding strategies

4.6.1 Simulation Setup

The simulations are based on a stationary WSN of size 200×200 m^2. All sensor nodes are uniformly distributed in this area, while one node is chosen at random as the sink. We simulated several *rounds* until all forwarding paths of the appropriate strategy were established and stable, applying the packet reception ratio and energy model presented in Section 4.3. As described in Section 4.3.4, we assume that all nodes will have already performed some kind of link measurements and know about the packet reception ratios on the wireless links to their neighbors. After establishing the forwarding paths, each node will generate data packets that will then be forwarded towards the sink, evaluating the end-to-end delivery ratio as well as the energy consumption.

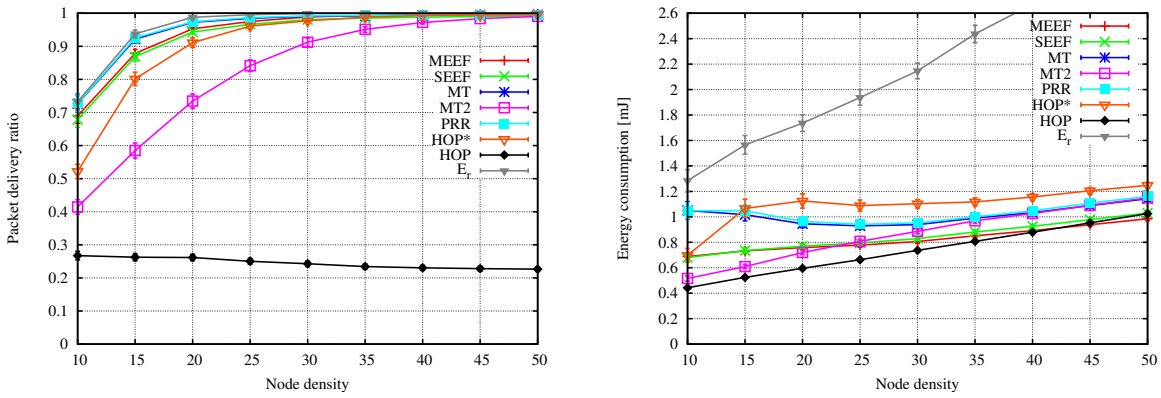
We have performed simulations in order to investigate the influence of single parameters, which included the node density, contention on forwarding links, the maximum number of retransmissions, the packet size of forwarding messages, and the energy cost of receiving messages. To obtain stable results, we performed 500 runs per simulation set. All graphs presented in the following depict average values over these runs and the corresponding 0.95 quantiles.

4.6.2 Influence of Node Density

At first, we investigate the influence of the node density μ for different performance aspects. The density is varied between 10 and 50 nodes per transmission range. The maximum number of retransmissions R is set to three; the size of data packets issued by source nodes, which is denoted by k, is set to 32 bytes.

Using this parameter set, Figure 4.11(a) depicts the end-to-end delivery ratio averaged over all nodes in the network. Among all single-link forwarding strategies, E_r-based forwarding performs best and gives an upper bound for the achievable delivery ratio. On the other hand, simple hop-based forwarding shows the worst results. Because blacklisting is not applied, most forwarding links are lossy, which of course leads to bad delivery ratios. As we have seen in Section 4.4.1, improving hop-based forwarding by blacklisting achieves significantly better results. It then even performs better than MT2 forwarding, which tries to minimize the *expected* number of transmissions on a forwarding path. However, PRR-based forwarding shows a better trade-off between the link quality and the length of the forwarding path. Although it does not consider the end-to-end delivery ratio explicitly, it performs quite well and almost reaches the packet delivery ratio of E_r-based forwarding. A similar performance is achieved by MT forwarding, where link qualities are implicitly considered. As MT forwarding assumes *infinite* retransmissions to calculate the expected number of required transmissions, links having poor reception ratios are downgraded more than in MT2 forwarding, which considers the *real* number of transmissions. For this reason, its delivery ratio is higher, higher even than for SEEF and MEEF. However, SEEF and MEEF try to find a trade-off between the delivery ratio and the corresponding energy consumption. As MEEF broadcasts data packets to a forwarding set rather than to a single node, it is able to improve the delivery ratio of SEEF without spending too much additional energy.

As shown in Figure 4.11(b), MEEF spends even less energy than almost any other strategy on a forwarding path for node densities larger than 25 (except for hop-based forwarding that suffers from



(a) Packet delivery ratio

(b) Energy consumption

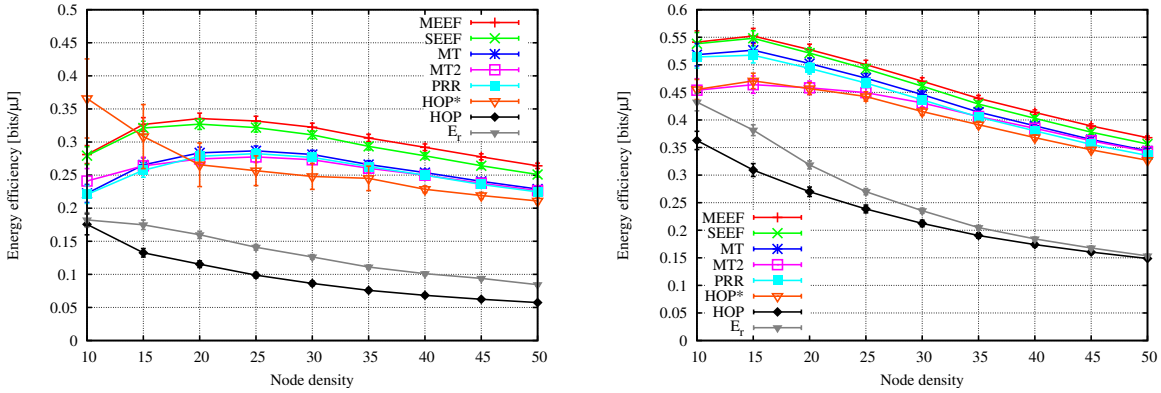
Figure 4.11: Packet delivery ratio and Energy consumption ($k = 32$, $R = 3$)

low delivery ratios and thus consumes less energy on end-to-end paths due to early packet drops). Although multi-link forwarding requires more nodes to receive a packet and in some cases requires polling messages, these additional energy costs can be compensated due to a better delivery ratio and thus fewer retransmissions. However, even if multi-link forwarding is not applied, the energy consumption of SEEF is significantly lower than that of the other forwarding strategies achieving a similar delivery ratio.

Most energy is consumed by E_r-based forwarding although it achieves the best delivery ratio, and retransmissions seldom occur. Thus, many nodes are involved in the forwarding process, which increases the total energy consumption, even if the energy cost per link might be low. On the other hand, hop-based forwarding "benefits" from its bad end-to-end delivery ratios because many packets do not reach the sink and so do not cause any further energy consumption. However, if the density increases, more nodes will originate data packets, with a greater likelihood of finding better paths with low hop counters. But since packet drops remain a dominant factor, the overall energy consumption will be comparably low. In contrast, hop*-based forwarding consumes more energy since more packets reach the sink. While for a density of 10 nodes, the energy consumption is lower due to disconnections caused by blacklisting, blacklisting in the case of higher densities leads to better connected forwarding paths. Since hop*-based forwarding reduces the number of packet drops significantly, more packets are finally delivered such that even more energy is needed. Moreover, because many retransmissions are still required, it also performs worse than any other strategy, except for E_r-based forwarding. Concerning the energy consumption of MT and PRR-based forwarding, an interesting effect can be observed: Up to a density of 25 nodes, the energy consumption decreases, although all other strategies consume more energy. We can explain this behavior as follows: If the node density is low, the number of well-connected neighbors is low, too. Achieving high delivery ratios thus requires much energy. Furthermore, finding short forwarding paths is unlikely. But if the density increases, the likelihood of finding well-connected forwarding nodes with low hop counters grows, decreasing the number of retransmissions and involved forwarders on a path. However, for higher densities, the energy consumption starts to grow again. Because more available neighbors increase the end-to-end delivery ratio, even on longer paths, packet drops are avoided in many cases. Thus, more energy is

consumed by forwarding. Additionally, if more nodes are deployed in the network, more nodes will be affected by packet transmissions and consume energy, at least to detect packet headers.

The energy efficiency of the different forwarding strategies is shown in Figure 4.12. While Figure 4.12(a) depicts the efficiency of the entire network, Figure 4.12(b) shows the energy efficiency per node. The difference is as follows. While the energy efficiency of the entire network is calculated from $8k \sum_i E_i^r / \sum_i E_i^e$, the efficiency per node is $\frac{8k}{N} \sum_i (E_i^r / E_i^e)$, where k denotes the data packet size and N the number of nodes in the network. For example, disconnected nodes have an efficiency of zero and thus are not considered in terms of the network efficiency but in terms of the energy efficiency per node. Furthermore, high variances in the nodes' energy efficiency have a deeper impact on the average per-node value. Due to this reason, the performance regarding the network efficiency and the efficiency per node might be different, especially for low node densities where network partitions are not unlikely. As hop*-based forwarding might cause further disconnections due to blacklisting, its network efficiency is even better than that of SEEF and MEEF for a low node density. However, as illustrated in Figure 4.12(b), the energy efficiency per node is clearly lower, as many nodes have zero efficiency.



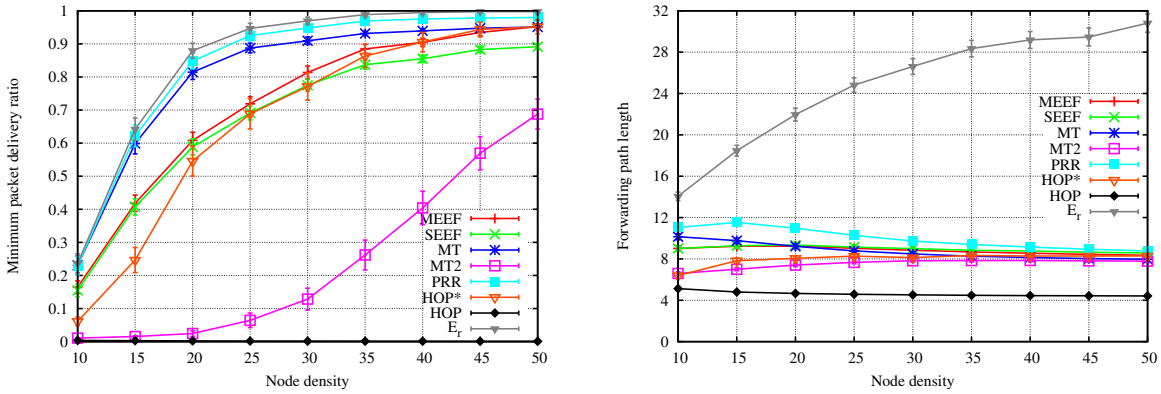
(a) Energy efficiency

(b) Energy efficiency per node

Figure 4.12: Energy efficiency and energy efficiency per node ($k = 32$, $R = 3$)

Thus, SEEF, as well as MEEF, improves the end-to-end energy efficiency substantially in almost all cases. Even if neither strategy achieves the highest delivery rates, they nonetheless trade off the ratio between delivered information and energy cost better than all other forwarding strategies. Once again we see that the performance of SEEF can be improved by multi-link forwarding, which achieves a better packet delivery ratio while at the same time conserving energy through fewer retransmissions. Among the other strategies, MT, MT2 and PRR-based forwarding perform quite similarly, particulary for moderate and high node densities. Even if none of these considers the end-to-end delivery ratio explicitly, the packet reception ratios on forwarding links are taken into account. Hence, hop*-based forwarding performs significantly better than simple hop-based forwarding without blacklisting. However, its efficiency is even worse than that of E_r-based forwarding, which heavily suffers from its high energy cost and thus performs badly concerning energy efficiency, although it achieves the best end-to-end delivery ratio.

Finally, Figure 4.13 illustrates the influence of the node density on the end-to-end delivery ratio with respect to the worst-connected node in the network, as well as to the average forwarding path length. As expected, E_r-based forwarding achieves the highest minimum delivery ratio, while hop-based forwarding performs worst. Like in Figure 4.11(a), PRR-based forwarding performs better than both MT strategies, hop*-based forwarding, and MEEF and SEEF. A comparison of MT and MT2 forwarding shows that MT benefits from its assumption of infinite retransmissions since bad links are downgraded more than in MT2 forwarding. Even the minimum delivery ratio of hop*-based forwarding is significantly better than that of MT2 because forwarding links with a packet reception ratio lower than the blacklisting threshold are not used. Although the performance of MEEF and SEEF is only moderate, it is sufficient, considering its higher efficiency.



(a) Minimum packet delivery ratio

(b) Forwarding path length

Figure 4.13: Minimum packet delivery ratio and forwarding path length ($k = 32$, $R = 3$)

Concerning the forwarding path length depicted in Figure 4.13(b), hop-based forwarding performs best, as it establishes the shortest path tree in the network. In contrast, the average path length of E_r-based forwarding seems to give an upper bound for all other strategies as it relies mainly on short-distance links. If hop-based forwarding is combined with blacklisting, the forwarding paths get longer. We also see that the forwarding paths of the MT2 strategy are shorter than those of MT. As mentioned above, this is due to the fact that MT2 forwarding sometimes prefers links that perform worse, because they are long-distance links. Like hop-based forwarding, the path lengths in PRR-based forwarding tend to decrease for higher node densities since it is more likely that short paths over well-connected neighbors will be used. The small increase at the beginning is again caused by the high number of disconnected nodes that vanish if more nodes are deployed. Considering the average path length of MEEF and SEEF, we see that multi-link forwarding slightly reduces the number of hops until packets have reached the sink. Thus, the forwarding set used by MEEF often contains nodes with a lower hop counter. Although such nodes are likely to have a worse packet reception ratio, the better delivery ratio of the entire forwarding set compensates this effect, as shown in Figure 4.11(a). Furthermore, due to the lower path lengths of MEEF, the number of packet transmissions can be reduced, too. This again explains its lower energy consumption compared to SEEF.

In conclusion, increasing the node density provides better links and improves the end-to-end delivery ratio. However, at the same time, more energy is consumed, as more nodes are affected by packet transmissions. Thus, the energy efficiency decreases if more nodes are deployed within the network.

Although SEEF already achieves substantial performance gains, the multi-link concept of MEEF further improves its energy efficiency due to a better packet delivery ratio and lower energy cost. Even if E_r-based, PRR-based, and MT forwarding perform better concerning the end-to-end delivery ratio, their energy consumption is significantly higher, which results in a worse energy efficiency.

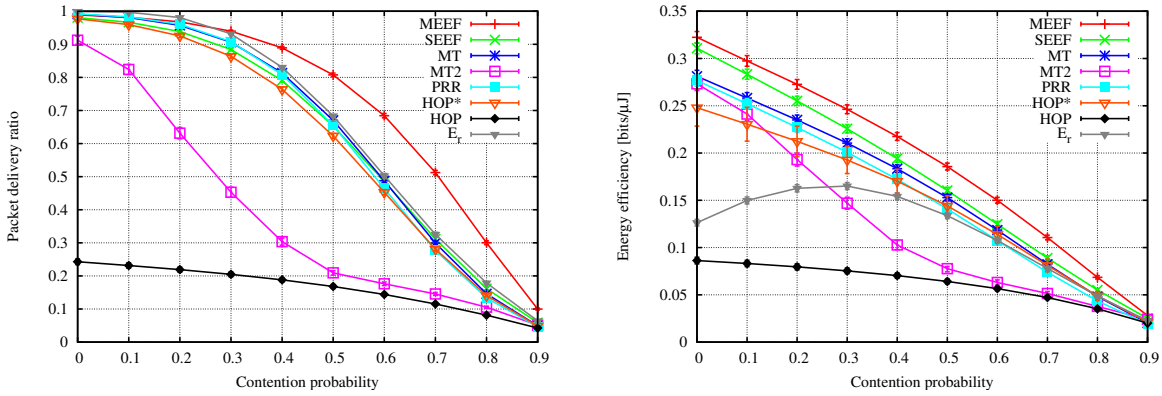
In the following sections, we now analyze the influences of several other conditions like contention, retransmissions, different packet sizes, and receiving energy cost.

4.6.3 Influence of Contention

As contention on forwarding links has so far not been considered, this section investigates the impact if several nodes try to access the wireless channel concurrently. As such interference is not covered by the packet reception model presented in Section 4.3.1, we extend the model as follows: Although contention usually relies on the number of nodes that intend to send a packet at the same time and thus may vary within the network, we will assume that it is the same for all nodes in the network. Given the contention probability ρ, the probability that a packet will be received correctly over a link (i, j) is then $prr_{i,j}(1 - \rho)$. Furthermore, the probability that a sender i will receive an acknowledgement successfully is $prr_{i,j}(1 - \rho)prr_{j,i}$, as acknowledgements are commonly sent directly after the end of the packet transmission. Like before, we assume that link measurements are available which evaluate the link qualities over time. Because each transmission affects the estimates, the amount of contention is implicitly captured. Thus, we must assume that nodes only have knowledge about packet reception ratios that are influenced by contention effects.

In order to investigate the influence of contention on the forwarding performance, the contention probability is increased from zero to one, and a density of 30 nodes is considered. Again, all nodes generate data packets that are forwarded towards the sink once a forwarding tree has been established. As before, the packet size of all forwarding messages is assumed to be 32 bytes. To compensate for higher contention probabilities, up to three retransmissions are used.

In Figure 4.14(a), the impact on the end-to-end packet delivery ratio is shown. For $\rho = 0$, the results are identical to the ones depicted in Figure 4.11(a). As contention is not taken into account at this point, all strategies (except for hop-based forwarding) achieve quite high delivery ratios. Like before, E_r-based forwarding achieves the best results. However, for $\rho > 0$, the benefit of exploiting multi-link forwarding increases. Because the presence of additional forwarders improves the probability of forwarding packets successfully, using MEEF is more advantageous. If the contention probability is higher than 0.25, it even outperforms E_r-based forwarding, which considers only single-link forwarding. Especially if the packet reception ratio on forwarding links is low, it will be more efficient to send packets to more than one forwarder and to compensate for the additional energy required by multi-link forwarding. The difference in the performances of SEEF and MEEF thus grows significantly for $\rho > 0$. While the delivery ratio of MEEF is just about 3% better than that of SEEF for $\rho = 0$, the improvement is about 60% for a contention probability of up to 0.7. However, for $\rho \to 1$, the gain of multi-link forwarding vanishes, as the packet delivery ratio drops to zero.



(a) Packet delivery ratio

(b) Energy efficiency

Figure 4.14: Packet delivery ratio and energy efficiency ($\mu = 30$, $k = 32$, $R = 3$)

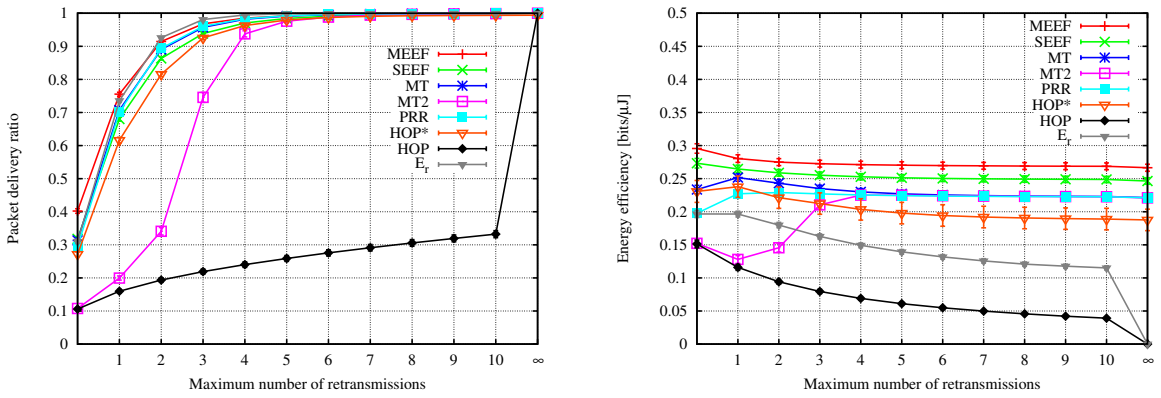
The corresponding network energy efficiency is depicted in Figure 4.14(b). Again, MEEF shows substantial improvements over SEEF, especially for moderate contention probabilities. The main reason is that it benefits from its higher packet delivery ratio and thus increases the number of delivered bits per energy unit. Moreover, due to its multi-link concept, it requires less energy per bit in order to forward packets towards the sink if the contention on forwarding links increases. But even if multi-link forwarding is not applied, the energy efficiency of SEEF is still significantly better than the efficiency of all other strategies.

While the relative performance by almost all strategies remains the same for different contention probabilities, MT2 and E_r-based forwarding show a different behavior: With an increasing contention probability, MT2 performs similar to hop-based forwarding since the majority of links have costs equal to the maximum number of retransmissions. On the other hand, E_r-based forwarding first benefits from contention because it reduces the number of hops on forwarding paths established. The number of transmissions and thus the overall energy consumption then decreases, which leads to a better energy efficiency. However, if the contention becomes too high, retransmissions will consume too much energy, again degrading the efficiency of E_r-based forwarding.

4.6.4 Influence of Retransmissions

With an increasing number of retransmissions, the end-to-end delivery ratio is expected to increase, too. However, as also more energy will be consumed, it is interesting how the energy efficiency is affected. For $R \to \infty$, we get $E_i^r \to 1$ from Equation 4.14 and Equation 4.19, respectively. That is, in this case, the energy efficiency is solely based on energy costs.

Figure 4.15(a) illustrates the packet delivery ratio if the maximum number of retransmissions is varied. For a density of 30 nodes and a packet size of 32 bytes, we simulated a retransmission range from zero to ten, and in addition the ∞-retransmission case. Furthermore, we set the contention probability to 0.2 in order to easier distinguish between different retransmission values.



(a) Packet delivery ratio

(b) Energy efficiency

Figure 4.15: Packet delivery ratio and energy efficiency ($\mu = 30$, $k = 32$, $\rho = 0.2$)

As expected, increasing the maximum number of retransmissions considerably improves the delivery ratio. Except for hop-based forwarding, all strategies achieve a packet delivery ratio above 0.9 if more than three retransmissions are used. For $R > 5$, the delivery ratio is almost perfect and shows no significant difference to the ∞-retransmission case. In contrast, hop-based forwarding heavily suffers from ignoring link qualities and achieves only a delivery ratio of 0.32, even if up to ten retransmissions are used.

The changes in the energy efficiency are shown in Figure 4.15(b). If the maximum number of retransmissions is increased, the energy efficiency starts to decrease. Since fewer packet drops then occur along forwarding paths, the total amount of consumed energy increases because more packets are finally delivered, slightly reducing the energy efficiency. For $R \to \infty$, the efficiency of hop-based forwarding drops to zero, as the energy cost grows more than the delivery ratio improves. Since the expected number of transmissions on a link is $(prr_{i,j}(1-\rho)prr_{j,i})^{-1}$ if the number of retransmissions is unlimited, note that even one bad link may lead to an efficiency of almost zero. Although not obvious, E_r-based forwarding suffers from the same effect. Due to its forwarding metric, using either good or bad links makes no difference because the end-to-end delivery ratio approaches one in any case. Thus, it is not unlikely that an incredible amount of energy will be spent on bad forwarding links.

Another interesting effect can be observed for MT2 forwarding if no retransmissions are used at all. As MT2 forwarding tries to minimize the expected number of transmissions that is calculated by $\frac{1-(1-prr_{i,j}prr_{j,i})^{R+1}}{prr_{i,j}prr_{j,i}}$, $R \to 0$ leads to hop-based forwarding, while $R \to \infty$ leads to the MT strategy. This change is clearly illustrated by the energy efficiency shown in Figure 4.15(b). Thus, for $R > 1$, the efficiency increases until it has reached the efficiency of MT.

The energy efficiency increase from $R = 0$ to $R = 1$ for MT, hop*-based and PRR-based forwarding can be explained by the high increase in the packet delivery ratio, which grows more than the energy consumption does. However, as the energy efficiency is not taken into account explicitly, low packet delivery ratios result in a worse efficiency. In contrast, if we consider the efficiency of MEEF and SEEF, we see that the performance gain of both strategies is significant and independent of the maxi-

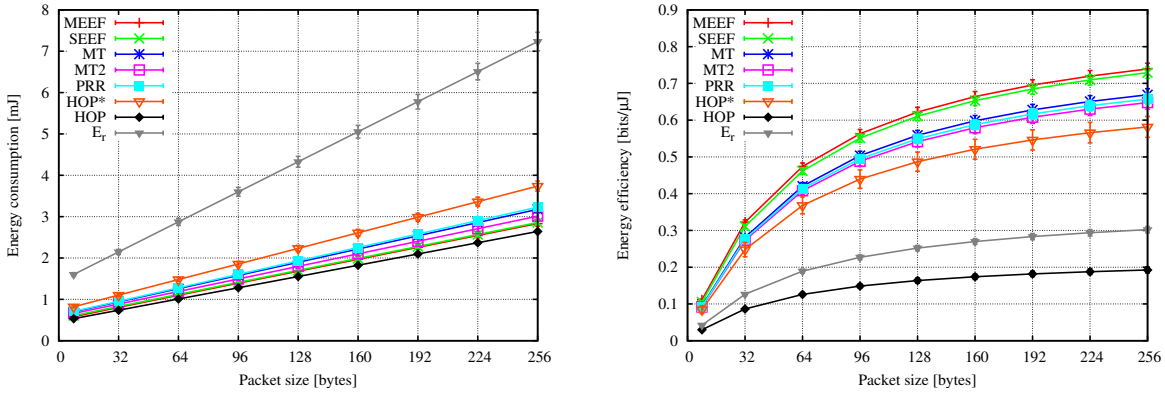
mum number of retransmissions used. Thus, both strategies again find the best trade-off between high delivery ratios and low energy costs.

4.6.5 Influence of Different Packet Sizes

In this section, we consider the influence of the packet size, which we have so far assumed to be 32 bytes. As before, we use a density of 30 nodes and set the maximum number of retransmissions to three. To be independent of the impact of contention, ρ is set to zero.

As the packet reception model presented in Section 4.3.1 does not take different packet sizes into account, the end-to-end packet delivery ratio is not affected in this simulation and is thus identical to the one shown in Figure 4.11(a). Extension of the model could be part of future work. Thus, we concentrate on the energy consumption and how it changes the energy efficiency.

Figure 4.16(a) depicts the average energy consumption if the size of the packets being forwarded is increased up to 256 bytes. Of course, the consumption will grow for larger packets, as transmitting and receiving consume more energy. However, the increase is proportional, since the expected number of transmissions will not change as long as larger packets do not influence the packet reception ratio. Thus, the relative improvement of MEEF and SEEF concerning the energy consumption remains unaffected, which is about 40% less than for E_r-based forwarding.



(a) Energy consumption

(b) Energy efficiency

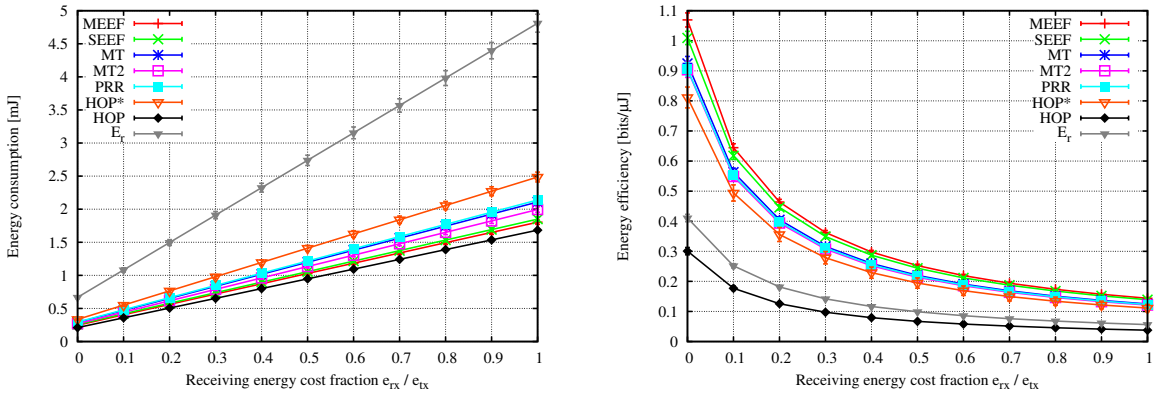
Figure 4.16: Energy consumption and energy efficiency ($\mu = 30$, $R = 3$, $\rho = 0$)

Because larger packets carry more data, the overhead per information unit is reduced, which is determined by acknowledgements, polling messages, and packet headers. For example, while the energy consumption of a packet with 64 bytes is less than twice as much compared to a size of 32 bytes, the number of delivered bytes doubles. Furthermore, since the delivery ratio remains constant, the energy efficiency is fully determined by the changes in the energy consumption. Thus, increasing the packet size leads to an asymptotical efficiency increase as shown in Figure 4.16(b).

4.6.6 Influence of Receiving Energy Cost

Finally, we consider the energy cost for receiving packets. As these costs are determined by the hardware used, it may be interesting to see how much the different forwarding strategies are influenced. So far, we have assumed an energy consumption of 35.84 μJ for transmitting and 12.9 μJ for receiving a 32-byte message, modeling the energy consumption of an ESB sensor node. However, using another radio transceiver might entail other energy costs. Thus, let us consider the impact of the receiving energy cost fraction e_{rx}/e_{tx} in more detail. We again consider a density of 30 nodes, a packet size of 32 bytes, and up to three retransmissions with no contention.

As in the previous section, the end-to-end packet delivery ratio is not affected by different cost fractions and thus does not change for this simulation. Figure 4.17(a) depicts the average energy consumption on a forwarding path by considering increasing energy cost fractions, i. e., by increasing e_{rx} from zero to e_{tx}. The corresponding energy efficiency is shown in Figure 4.17(b).



(a) Energy consumption

(b) Energy efficiency

Figure 4.17: Energy consumption and energy efficiency ($\mu = 30$, $k = 32$, $R = 3$, $\rho = 0$)

If $e_{rx}/e_{tx} \to 1$, the total amount of energy grows, as each transmission causes additional energy costs in terms of receiving. As in the last section, the relative performance over all forwarding strategies does not change, leading to results similar to those in Figure 4.16(a). However, since the number of delivered information units is unaffected, the increase of the energy consumption leads to a decreasing energy efficiency, as shown in Figure 4.17(b).

Except for MEEF and SEEF, changing the fraction e_{rx}/e_{tx} does not influence the forwarding metrics of the considered strategies. As a consequence, differences in the energy costs have no influence on their forwarding paths. However, if multi-link forwarding is considered, lower energy costs for receiving packets favor larger forwarding sets and thus increase the packet delivery ratio without spending much additional energy. Due to this reason, the relative performance improvement of MEEF over SEEF is higher if e_{rx}, rather than e_{tx}, is close to zero. Thus, the smaller the energy fraction e_{rx}/e_{tx}, the better the energy efficiency and the higher the gain of multi-link forwarding.

4.7 Experimental Evaluation

In addition to the performance results obtained by simulations, we have evaluated the considered forwarding strategies in our WSN testbed. Although the network is small in size and the number of nodes is considerably smaller than for the simulations, the results yield important indications about the performance in a real network. They also give a first proof-of-concept that all algorithms work on real sensor nodes.

4.7.1 ESB Implementation

We implemented all algorithms in ANSI C by means of the ScatterWeb firmware v2.3, which can be found under [5]. Software developed for the ESB platform mainly consists of two parts: the ESB firmware, which is, among other things, responsible for accessing the sensors of the platform and for the wireless communication, and an application, which contains the application logic of the sensor network. As shown in Figure 4.18, the EEF software is implemented as a separate module that lays between firmware and application. The main parts of the EEF module are the beacon module, the neighbor management, and the EEF handler. The beacon module triggers the periodic transmission of beacons in order to exchange forwarding and neighborhood information. By means of those beacons, the neighbor management updates the state information of each neighbor discovered, like the packet reception ratio, energy consumption, and packet delivery ratio. Using this information, the EEF handler calculates the best energy-efficient forwarding path towards the network sink and determines to which nodes packets need to be forwarded later on.

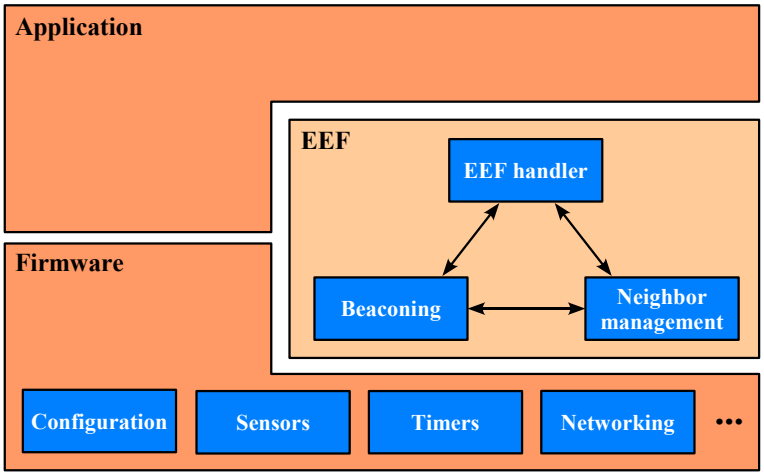


Figure 4.18: Software architecture of the ESB implementation

Because the content of beacons depends on the forwarding strategy considered, the structure of beacons is slightly different if different strategies are evaluated. As an example, Figure 4.19 illustrates the packet structure of SEEF and MEEF beacons. Besides the standard header, beacons contain the sink identifier and a node's forwarding metrics, i. e., the end-to-end delivery ratio E_i^r and the energy consumption E_i^e. The energy efficiency can later be calculated accordingly. In addition, the packet reception rates on the backward links between the node and its neighbors are included. If more than one

sink is deployed in the network, either forwarding paths to the best reachable sink or to all sinks may be maintained. In the latter case, the packet structure needs to be extended such that both forwarding metrics (E_i^r and E_i^e) are added per sink.

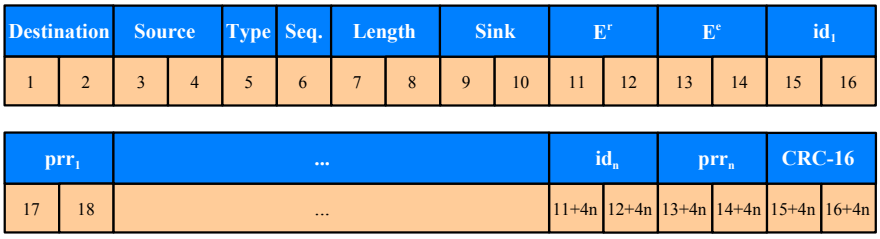


Figure 4.19: Packet structure of beacons

According to the information contained in a beacon, the neighbor management stores for each adjacent node its id, both E_i^r and E_i^e values, and the packet reception ratio regarding each link direction. Thus, a neighbor entry needs 10 bytes, which leads to a static neighbor table of 240 bytes for our experimental setup. In addition, the EEF module needs about 300 bytes to store statistics and to temporarily hold variables in order to calculate the node's energy efficient. Therefore, the EEF handler analyzes every neighbor and calculates the node'e end-to-end packet delivery ratio and energy consumption according to Equations 4.19 and 4.21. To speed up the computation, the size of a forwarding set is bounded to three, according to the results of Section 4.5.5.

Besides the packet structure of beacons, Figure 4.20 shows the structure of forwarding messages. In addition to the standard header, the sink and the identifier of the issuing node are included. In case multi-link forwarding is employed, a second and possibly even a third forwarding node may be specified. Next, the payload of the actual data follows, ending with the standard CRC checksum.

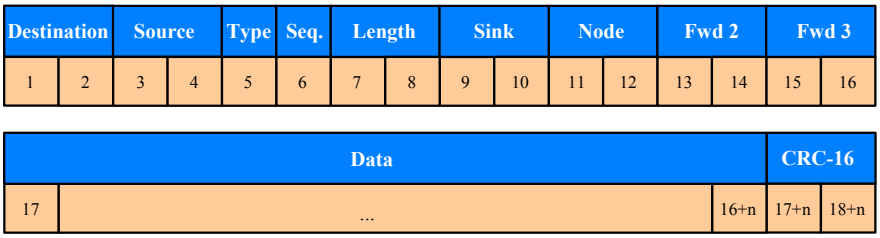


Figure 4.20: Packet structure of forwarding packets

4.7.2 Experimental Setup

The experimental evaluations are based on the testbed we used earlier in Chapter 3, which consists of 24 ESB nodes placed on a 4×6 grid and one additional node placed outside that acts as the sink. The nodes' transmission power is set to 15% in order to limit the range of communication, which results in a multi-hop network that relies on perfect as well as on lossy links while it is still fully connected.

According to the strategy being evaluated, the sink triggers the establishment of the forwarding tree. Once all paths are stable, each node issues 200 data packets over 100 minutes at a rate of one packet

per 30 seconds. The size of the node's reception buffer is set to 300 bytes. The data length is set to 32 bytes. Packet losses are taken into account by means of retransmissions; the maximum number is set to three. During the entire experiments, statistical information is captured at each sensor node and stored in the EEPROM for later processing.

In order to get first estimates about link losses, we run a "pinging process" initially: A node broadcasts 100 ping packets to its neighborhood, while all other nodes listen to the channel and log the number of properly received packets. Using that information, each node builds up a neighbor table containing the packet reception rates on forward and backward links, respectively. By including the forward link packet reception ratios into beacons, adjacent nodes learn about the appropriate backward reception ratios and can use them according to their forwarding metric.

After pinging, we start the actual evaluation of a specific forwarding strategy. Once no more data packets are issued, the sink will gather all relevant information by polling each node in a round-robin fashion and transmitting the results received over a serial link to a connected computer, which performs the statistical evaluation.

While evaluation metrics like the end-to-end delivery ratio or the number of hops a forwarding packet travels over can easily be calculated, measuring the exact power consumption of a node is non-trivial. Determining the consumption according to the decrease in the voltage of the node's batteries is often imprecise since good batteries commonly provide a high voltage over a long period of time before the voltage drops abruptly at the end of the battery lifetime. Exact results can be obtained only by measuring the amperage over time with an amperemeter. While this may be possible for a single sensor node, the overhead in a larger network would enormous and impractical for our WSN testbed.

We thus estimate the energy consumption by counting the number of packet transmissions and receptions, distinguishing between data and control packets. The energy consumption is then estimated according to the energy model presented in Section 4.3.3. Although this is a rough estimate, it should provide some indication about the magnitude of the real power consumption.

4.7.3 Evaluation Results

The experimental results are presented in Table 4.1. In each experiment, we evaluated a different forwarding strategy and logged information concerning the number of packets received and transmitted, the number of packets delivered to the sink, the energy consumption and efficiency, as well as the average number of traveled hops and required retransmissions. In order to slightly minimize variations in the measurements, we repeated each experiment ten times and computed average values together with the appropriate 0.95 t-quantiles.

The first four rows of Table 4.1 show the number of transmitted (tx) and received (rx) data, respectively control packets averaged over all nodes in the network. Comparing MEEF and SEEF indicates that MEEF sends slightly fewer data packets but more control packets than SEEF. Multi-link forwarding is thus really applied and keeps the number of retransmitted data packets to a minimum, at the expense of a higher control overhead due to polling. Since the multi-link concept may require more than one node to receive a data packet completely, the average number of received data packets is larger than for

Strategy	MEEF	SEEF	PRR
Tx data	1043.56 [980.51, 1106.61]	1071.93 [999.59, 1144.26]	1538.34 [1383.27, 1693.42]
Tx control	419.67 [329.80, 509.55]	386.45 [303.42, 469.47]	702.08 [591.57, 812.59]
Rx data	465.64 [390.42, 540.86]	462.67 [379.78, 545.55]	739.60 [634.94, 844.25]
Rx control	650.23 [561.85, 738.60]	553.14 [468.64, 637.65]	934.14 [815.09, 1053.19]
Packets delivered	3408.90 [3025.57, 3792.23]	3281.40 [2885.78, 3677.02]	3742.40 [3470.37, 4014.43]
Packets delivered per node	142.04 [126.07, 158.01]	136.72 [120.24, 153.21]	155.93 [144.60, 167.27]
Min. packets delivered	70.10 [59.04, 81.16]	67.20 [56.91, 77.49]	74.90 [61.55, 88.25]
Energy consumption [mJ]	3253.02 [3046.70, 3459.34]	3305.29 [3075.66, 3534.92]	4784.44 [4303.43, 5265.44]
Energy consumption per node [mJ]	135.54 [126.95, 144.14]	137.72 [128.15, 147.29]	199.35 [179.31, 219.39]
Max. energy consumption [mJ]	250.83 [219.08, 282.57]	264.18 [225.27, 303.09]	738.90 [536.72, 941.09]
Energy efficiency [bits/μJ]	0.2717 [0.2290, 0.3144]	0.2576 [0.2152, 0.3000]	0.2045 [0.1755, 0.2336]
Energy efficiency per node [bits/μJ]	0.3046 [0.2677, 0.3414]	0.2890 [0.2528, 0.3253]	0.2689 [0.2374, 0.3004]
Hop counter	2.76 [2.41, 3.11]	2.86 [2.45, 3.28]	3.38 [3.05, 3.72]
Retransmissions	0.82 [0.65, 0.98]	0.86 [0.70, 1.03]	0.73 [0.58, 0.88]

Strategy	E_r	HOP	HOP*
Tx data	2009.25 [1902.81, 2115.69]	1083.77 [978.74, 1188.80]	1192.59 [1117.01, 1268.16]
Tx control	907.12 [748.70, 1065.54]	172.79 [147.87, 197.71]	279.86 [248.38, 311.34]
Rx data	1001.66 [885.01, 1118.31]	269.23 [207.47, 330.99]	404.77 [350.32, 459.22]
Rx control	1228.60 [1067.96, 1389.24]	308.65 [259.52, 357.79]	444.92 [379.19, 510.64]
Packets delivered	3897.60 [3608.22, 4186.98]	2278.00 [1704.46, 2851.54]	2744.80 [2261.35, 3228.25]
Packets delivered per node	162.40 [150.34, 174.46]	94.92 [71.02, 118.81]	114.37 [94.22, 134.51]
Min. packets delivered	74.00 [62.24, 85.76]	6.20 [2.68, 9.72]	6.00 [0.13, 11.87]
Energy consumption [mJ]	6251.51 [5913.60, 6589.41]	3268.69 [2957.19, 3580.20]	3617.63 [3386.19, 3849.07]
Energy consumption per node [mJ]	260.48 [246.40, 274.56]	136.20 [123.22, 149.17]	150.73 [141.09, 160.38]
Max. energy consumption [mJ]	753.54 [656.34, 850.73]	384.23 [299.03, 469.42]	471.06 [294.17, 647.95]
Energy efficiency [bits/μJ]	0.1606 [0.1444, 0.1768]	0.1839 [0.1280, 0.2397]	0.1959 [0.1576, 0.2342]
Energy efficiency per node [bits/μJ]	0.2281 [0.2066, 0.2496]	0.2277 [0.1837, 0.2716]	0.2395 [0.1999, 0.2791]
Hop counter	3.50 [3.29, 3.72]	1.69 [1.63, 1.74]	2.10 [2.00, 2.20]
Retransmissions	0.87 [0.65, 1.08]	1.90 [1.79, 2.01]	1.49 [1.40, 1.58]

Strategy	MT	MT2
Tx data	1522.13 [1391.22, 1653.04]	1009.62 [927.89, 1091.35]
Tx control	693.39 [643.54, 743.24]	221.16 [186.21, 256.11]
Rx data	745.88 [690.15, 801.61]	276.20 [237.22, 315.18]
Rx control	918.09 [881.35, 954.83]	366.74 [318.99, 414.49]
Packets delivered	3715.90 [3456.43, 3975.37]	2598.20 [2111.85, 3084.55]
Packets delivered per node	154.83 [144.02, 165.64]	108.26 [87.99, 128.52]
Min. packets delivered	69.50 [56.49, 82.51]	11.50 [5.57, 17.43]
Energy consumption [mJ]	4732.21 [4341.23, 5123.18]	3068.17 [2821.38, 3314.95]
Energy consumption per node [mJ]	197.18 [180.88, 213.47]	127.84 [117.56, 138.12]
Max. energy consumption [mJ]	692.15 [552.22, 832.09]	316.84 [231.18, 402.49]
Energy efficiency [bits/μJ]	0.2054 [0.1737, 0.2371]	0.2212 [0.1683, 0.2741]
Energy efficiency per node [bits/μJ]	0.2672 [0.2350, 0.2994]	0.2520 [0.2109, 0.2931]
Hop counter	3.32 [3.08, 3.57]	1.99 [1.83, 2.15]
Retransmissions	0.70 [0.60, 0.81]	1.40 [1.30, 1.51]

Table 4.1: Results of the experimental evaluation (EEF)

SEEF. The same applies to the number of received control packets. However, the additional overhead is compensated by fewer retransmissions. As long as data packets consume more energy than do control packets, multi-link forwarding is more efficient. An additional advantage is that the number of delivered packets increases compared to SEEF[5]. As Table 4.1 illustrates, the delivery ratio of the worst connected node in the network is also slightly higher than the one achieved by SEEF. Thus, due to more delivered packets and a lower total energy consumption, MEEF performs better in terms of the network energy efficiency and the energy efficiency per node. The efficiency of SEEF is still significantly better than that achieved by other single-link strategies. Finally, the fact that multi-link forwarding is able to send packets faster towards the sink is shown in the last two rows of Table 4.1. As "backup" nodes are normally closer to the sink than the first node in a forwarder set, the average number of hops a packet has to take is slightly smaller. At the same time, additional forwarding nodes reduce the number of retransmissions, which in turn saves energy.

The performance of PRR-based forwarding is quite similar to that of MT and MT2. This confirms what we saw already in Section 4.6. However, MT2 performs slightly worse because its forwarding metric limits the maximum link cost per node to $R + 1$, according to the upper number of retransmissions. Bad links are thus penalized less than in MT forwarding, which is shown by the low number of delivered data packets, smaller hop counters, and more packet retransmissions. E_r-based forwarding again achieves the best packet delivery ratio, but at the expense of long forwarding paths and numerous packet transmissions. The energy consumption is thus very high in practice, almost twice the consumption of MEEF, which downgrades its efficiency considerably.

Interestingly, hop-based forwarding performs better than expected and simulated. This might be due to the fact that an initial pinging was not performed in the simulations; instead, it was assumed that all nodes knew about the exact packet reception ratio to adjacent nodes a priori. However, if the reception ratio is estimated according to the number of received ping packets, deviations are likely. It may therefore happen that sometimes not a single ping packet is received, particularly over bad links. In hop-based forwarding, this is even beneficial, as such neighbors are implicitly "blocked". In addition, the low node density and the small network size may reduce the number of available links and thus the number of forwarding alternatives. Nevertheless, the experimental results clearly show that hop-based forwarding performs worst in terms of the number of delivered packets. Due to its shortest path tree, it has the smallest hop counter per delivered packet, but causes the most retransmissions. That confirms our simulation results, as well as the fact that long-distance links often suffer from heavy packet losses. Hop*-based forwarding (with a blacklisting threshold of 0.1) slightly reduces this effect, which is shown by a larger hop counter and fewer retransmissions. As a consequence, the number of delivered packets and also the energy efficiency increase.

In conclusion, the experimental evaluation shows results quite similar to those we obtained in our simulations, although our testbed consisted of only 25 nodes. The results give us a first proof-of-concept that our energy-efficient forwarding is a promising approach to finding a good trade-off between the number of delivered packets and the required energy. Furthermore, the experiments confirm that using the concept of multi-link forwarding in conjunction with polling is a practical and at the same time efficient solution that could also be applied in other contexts.

[5]Note that each node has issued 200 data packet, i. e., the sink might receive up to 4,000 packets.

4.8 Conclusions

In this chapter, we have described different forwarding strategies for many-to-one communication in sensor networks where several sensor nodes report data to one predefined sink. Such data can either be periodically queried by the sink or generated according to a sensed stimulus in a node's vicinity. In both cases, data packets need to be routed through the network, as direct communication with the sink is often not possible. However, as our results have shown, there is a trade-off between maximizing the end-to-end delivery ratio and the energy consumption.

We have proposed two forwarding strategies called SEEF and MEEF that seek to maximize the energy efficiency of forwarding paths, which we defined by the ratio of delivered data per energy unit. While the performance of SEEF always gives us a lower bound, MEEF achieves further improvements by multi-link forwarding. Instead of considering just a single forwarder, it exploits the broadcast characteristics of the wireless channel more efficiently and may use several links in order to forward a packet. Despite a higher overhead caused by additional control packets, retransmissions of data packets can often be avoided, which in turn reduces the overall energy consumption.

We have evaluated the energy efficiency of SEEF and MEEF, both by simulation and by implementation and testing; both strategies show the best performance among a variety of forwarding schemes proposed in the literature. Furthermore, as the simulation as well as the experimental results have shown, in the majority of cases MEEF also improves the end-to-end delivery ratio, increasing the energy efficiency along a forwarding path.

Extending SEEF and MEEF to the multi-sink case can easily be achieved by calculating and propagating the energy efficiency of a node for each sink separatively. In case data packets may later be forwarded to an arbitrary sink in the network, the best one in terms of energy efficiency is selected. Otherwise, data packets are sent to the sink that requested them. Thus, the number of sinks does only influence the computation time and the memory usage of the EEF algorithm.

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

- the analysis to what degree hop-based and PRR-based forwarding can be improved by blacklisting badly connected neighbors,
- the proposal of two new forwarding strategies that aim at maximizing energy efficiency called *multi-link and single-link energy-efficient forwarding*,
- the general concept of multi-link forwarding that achieves a better energy efficiency than single-link schemes,
- mathematical derivations for calculating the average end-to-end packet delivery ratio and energy consumption of forwarding; both are used to compute the energy efficiency concerning the case of infinite as well as finite retransmissions;
- simulations that present the influence of different network characteristics on the forwarding performance, like the node density, network contention, and the number of retransmissions,

- real-world experiments using a WSN testbed consisting of 25 ESB nodes that provide a proof-of-concept and indicate how forwarding schemes work in practice.

So far, the residual energy of forwarding nodes has not been taken into account. A problem that arises from using MEEF or SEEF is that forwarding paths will not change as long as the end-to-end packet delivery ratio as well as the energy consumption are not affected. Thus, nodes along the most energy-efficient paths are used permanently until they run out of energy. This could be problematic if such nodes are required for special application tasks. In the worst case, network partitions might even occur.

In the following, we thus extend MEEF and SEEF by a *network lifetime* component that takes into account the residual energy along a forwarding path. The energy consumption throughout the network should then be better balanced, increasing the lifetime of frequently used nodes. However, energy efficiency must not be neglected since solely maximizing the lifetime of the entire network might lead to worse performance results, as illustrated in the following chapter.

CHAPTER 5

Lifetime-Efficient Forwarding

"Being abstract is something profoundly different from being vague. The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

– E. Dijkstra –

5.1 Introduction

Commonly, routing algorithms that aim at minimizing energy costs to extend the lifetime of a network employ shortest path algorithms, with the energy required for the transmission between two adjacent nodes used as the edge cost. However, as the results of the last chapter have shown, minimizing the energy costs might negatively affect the data delivery ratio. Thus, forward algorithms that cause low energy costs but high loss rates are often not efficient. In contrast, focusing on energy efficiency turned out to be a very promising forwarding scheme. It trades off the end-to-end delivery ratio of nodes reporting data to a sink node and the energy consumption in the network very well. In this way, it considerably outperforms strategies based solely on either the energy consumption or on delivery ratios.

However, even if energy-efficient forwarding is employed, the overall lifetime of the network need not be maximized. Once forwarding paths have been determined, they will remain stable as long as their costs do not change. Thus, nodes along optimal paths are used quite often and consume more energy than others if the traffic load is not balanced within the network. As such nodes quickly run out of energy, the likelihood of broken paths or network partitions increases. This problem is addressed by *maximum lifetime algorithms* designed to prevent the early "burn out" of such paths, focusing either on the lifetime of single nodes or of the entire network.

As the definition of lifetime is not always the same, it is quite difficult to compare different approaches. Lifetime is often defined as the time until the first node in the network runs out of energy [23, 28], the time until the entire network energy has been consumed, or as the time at which the first network

partition occurs. Definitions referring to the time until at least a fraction of α nodes are alive or at least an α coverage is maintained [273] can also be found. However, most work uses the time until the first node "dies", as this definition can easily be implemented by linear programs tackling the maximum lifetime problem.

Compared to existing work focusing on the maximum lifetime problem, we emphasize the existence of packet losses and propose forwarding strategies that avoid lossy links, which might cause a high energy consumption due to retransmissions. Most of the work in the literature assumes that two nodes will always be able to communicate with each other as long as their distance is lower than the maximum radio range. However, real-world measurements have shown that this is not always true because many regions within the communication area show high variations. Asymmetric links often occur, which might have a deep impact on the delivery of acknowledgements and thus on the network lifetime if much energy is consumed along lossy forwarding paths.

Furthermore, relying on maximizing the lifetime need not lead to good delivery ratios or a low energy consumption if *neither* metric is not taken explicitly into account. For example, in terms of the total energy consumption within the network, it may sometimes even be better to use forwarding paths with high loss rates. In this case, packets sent along a forwarding path may be dropped early such that no further energy will be consumed. Forwarding data over such paths might thus be "good" if only the energy consumption or the network lifetime is considered. But concerning the packet delivery ratio it is contradictory and senseless. Thus, we must keep this trade-off in mind.

The remainder of this chapter is organized as follows: In the next section, we will outline related work and present existing energy-aware algorithms. Section 5.3 then describes the maximum lifetime problem based on a linear programming formulation and provides an option for solving the problem heuristically. Based on the energy-efficient forwarding strategies we considered in Chapter 4, Section 5.4 presents an extension of MEEF and SEEF that takes into account the residual energy (and thus the node's lifetime), which we will call *lifetime-efficient* forwarding. By means of simulations, Section 5.6 analyzes the performance of lifetime-efficient forwarding with regard to different metrics and compares it to other approaches. Results from real-world experiments are presented in Section 5.7. Finally, Section 5.8 concludes the chapter and summarizes its main contributions.

5.2 Related Work

Addressing energy consumption in wireless sensor networks has been an active research field [176, 196]. There exists much work on power control algorithms, whose goal is to find an optimal transmission power for each node without the loss of connectivity [82, 118, 157, 197, 204, 246]. While maintaining connectivity, the resulting topology may aim at, e. g., improving the network throughput [82, 118], bounding the number of 1-hop neighbors [197], or minimizing the overall power consumption required for routing [157, 204, 246]. Lloyd *et al.* [157] study the algorithmic aspects of minimizing the total energy consumption, [204] and [246] present topology control algorithms that focus on the distributed nature of sensor networks. Further approaches are described by Li in [154], which also provides a fundamental survey on topology control.

On the other hand, several approaches exist that are not designed for topology control but for *power-aware* or *energy-aware routing*, aiming at either minimum energy routing or maximum network lifetime routing. The former approach tries to minimize the total energy required in order for a packet to reach its destination. However, nodes along the optimal path are used very frequently and soon run out of energy, likely causing network partitions although many nodes might still have enough energy [218]. Furthermore, minimum energy routing causes an imbalance of power consumption. This issue is addressed by algorithms that are designed to maximize the network lifetime.

While classic routing protocols are based on minimum hop routing [124, 126, 179, 185, 190], power-aware algorithms are rather based on power-based metrics that are used in combination with a shortest path algorithm [225, 229, 238, 153]. Singh *et al.* propose several metrics for power-aware routing in [225], including cost per packet and node, time until the network gets partitioned, consumed energy per packet, and variance in residual energy. In [229], a localized algorithm is proposed by Stojmenovic and Lin. Based on a node's lifetime and distance-based power metrics, the algorithm aims to extend its worst-case lifetime. In [238], Tho presents a conditional max-min battery capacity routing algorithm that combines minimum total energy routing and max-min residual energy routing: if the minimum residual energy along a path is higher than a given threshold, minimum total energy routing will be performed. Otherwise, the forwarding path is selected such that the minimum residual energy of a node on the path is maximized. In [153], the authors study the problem of the maximum network lifetime, assuming an unknown message flow. They propose an approximation algorithm called $\max - \min zP_{min}$ that tries to trade off between minimum transmission energy and max-min residual energy routing. First, the algorithm determines the path with the lowest transmission costs, with P_{min} being its required transmission energy and r being the node's minimum residual energy on this path. Then, all edges whose residual energy is less than or equal to r are removed from the graph. This procedure is repeated until the total transmission energy of the minimum transmission energy path exceeds z times P_{min}, where $z \geq 1$. Li *et al.* have shown that their approximation performs well and is close to the optimal solution they obtained by linear programming. Other power-aware algorithms can be found, e. g., in [76, 100, 175, 177, 194]

The works in [52, 129, 208, 263] also formulate the problem of the maximum network lifetime by using linear programming. Chang and Tassiulas [52] present a heuristic algorithm to solve the linear program approximately. They consider two different models for the information-generating process: The first assumes constant rates and the second assumes an arbitrary generating process. An approximation to the problem based on the Garg-Konemann [94] algorithm achieves a $(1-3\epsilon)$-approximation to the optimal network lifetime for any $\epsilon > 0$ [53]. While the heuristic algorithm of Kalpakis *et al.* [129] improves this approximation to $(1-\epsilon)$, their algorithm does not scale very well with the network size. Achieving the same approximation, Xue *et al.* [263] present an algorithm with a better runtime: they consider the possibility of data aggregation, and in doing so, improve the runtime of the fastest existing algorithm by a factor of K. K denotes the number of commodities which represents the data generated by a sensor node and delivered to a destination. Sankar and Liu [208] finally formulate the problem as a maximum flow problem and adapt distributed flow algorithms [17, 18].

In contrast to existing work, which mainly optimizes the network's lifetime only, we rather emphasize on energy efficiency and lifetime at the same time. Our goal is to maximize both metrics simultaneously, i. e., each node selects a forwarding path that maximizes the product of the expected end-to-end

energy efficiency and the expected minimum residual energy on that path. In so doing, it is possible to find a good trade-off between energy consumption, packet delivery performance, and network lifetime.

5.3 The Maximum Lifetime Problem

In order to compare the network lifetime of different forwarding strategies with the maximum possible network lifetime, we consider two linear programs (LP). While the first one tackles the maximum lifetime problem if the number of packet retransmissions is finite, the second one assumes an infinite number of retransmissions. We use the following definitions: Let $G(N, A)$ be a directed graph, with N being the set of nodes and A being the set of edges. If there exists a directed link between two nodes i and j, then $(i, j) \in A$, with prr_{ij} denoting the packet reception ratio on that link. Regarding this graph, let N_i be the set of neighbors of i and s be the sink node. Concerning the energy consumption, e_{rx}, e^h_{rx}, and e^a_{rx} specify the amount of energy units required to receive an entire packet, only the packet header, or an acknowledgement. The energy consumption for transmitting a data packet or an acknowledgement is specified as e_{tx} and e^a_{tx}, respectively. Furthermore, let E_i denote the initial energy of a node i. Then, T refers to the maximum lifetime of the entire network, defined as the time until the first node runs out of energy.

5.3.1 Finite Retransmissions

If we assume that all nodes are connected and that each node sends a data packet to the sink every lifetime unit (round), we can formulate the maximum lifetime problem as a linear program, with t_{ij} being the number of packets sent by node i to node j over its entire lifetime. Solving the LP will then yield optimal values t^*_{ij} that maximize T.

Due to a finite number of retransmissions, it is possible that not all packets will reach the network sink because packet losses might occur during transmissions. If we denote the maximum number of retransmissions as R and assume that each data packet will be resent until an acknowledgement is received, we can calculate delivery probabilities ρ_{ij} by $1 - (1 - prr_{ij})^{R+1}$ and ρ^a_{ij} by $1 - (1 - prr_{ij}prr_{ji})^{R+1}$. While ρ_{ij} refers to the probability that a packet sent over a link (i, j) by node i will be correctly received by node j, ρ^a_{ij} denotes the probability that the packet will, in addition, be correctly acknowledged. Then, the expected number of transmissions is

$$\tau_{ij} = \begin{cases} R & prr_{ij} = 0 \vee prr_{ji} = 0, \\ \frac{\rho^a_{ij}}{prr_{ij}prr_{ji}} & \text{otherwise.} \end{cases} \tag{5.1}$$

The corresponding number of expected acknowledgements sent by node j is $\tau^a_{ij} = prr_{ij}\tau_{ij}$.

Using these definitions, the LP is of the following form:

$$\textbf{LP 1:} \quad T \rightarrow \max \tag{5.2}$$

s. t.

$$\sum_{j \in N_i} t_{ji}(\tau_{ji} e_{rx} + \tau^a_{ji} e^a_{tx}) + \sum_{j \in N_i} \sum_{k \in N_j, k \neq i} t_{jk} \tau_{jk} e^h_{rx} + \sum_{j \in N_i} t_{ij}(\tau_{ij} e_{tx} + \tau^a_{ij} e^a_{rx}) \leq E_i, \qquad \forall i \in N \backslash \{s\} \tag{5.3}$$

$$\sum_{j \in N_i} t_{ij} = \sum_{j \in N_i} t_{ji} \rho_{ji} + T, \qquad \forall i \in N \backslash \{s\} \tag{5.4}$$

$$u_i - u_j \geq 1 + (x_{ij} - 1)|N|, \qquad \forall i \in N, j \in N_i \tag{5.5}$$

$$t_{ij} \leq M x_{ij} \qquad \forall i \in N, j \in N_i \tag{5.6}$$

with x_{ij} being a binary variable, u_{ij} an integer variable, and M a sufficiently big number.

Equation 5.3 covers the energy constraint and takes into account the energy consumption of the transmission and reception of data packets, respectively acknowledgements. The sink node is assumed to have infinite energy. Equation 5.4 refers to the forwarding constraint, which specifies that for each node the number of outgoing packets is equal to the number of incoming packets plus the number of originated packets at this node.

In order to ensure that all packets are sent *towards* the sink, routing cycles need to be prohibited, which is considered by Equation 5.5 and 5.6 as follows: Each node i on an arbitrary forwarding path is assigned a number u_i such that for each link $i \rightarrow j$ the constraint

$$u_i - u_j \geq 1 \tag{5.7}$$

is satisfied. Nodes that are not connected by a forwarding link, i. e., a link with $t_{ij} > 0$, are not restricted. Since there are at most N nodes in the graph, it is sufficient to allow

$$u_i - u_j \geq 1 - N \tag{5.8}$$

for such links. Both constraints are combined in Equation 5.5, where x_{ij} is defined as

$$x_{ij} = \begin{cases} 1 & t_{ij} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

By using a sufficiently large number M, this definition can be mapped to the linear constraint shown in Equation 5.6.

5.3.2 Infinite Retransmissions

If the number of retransmissions is infinite, the problem of routing cycles can be taken into account more easily by exploiting the fact that as long as prr_{ij} is greater than zero, any packet sent by node i will eventually reach node j, even if it may take a long time. Thus, it is sufficient to require that each

data packet issued in the network be received by the sink. The constraints in Equations 5.5 and 5.6 are no longer needed and can be simplified as follows:

$$\textbf{LP 2:} \quad T \rightarrow \max \tag{5.10}$$

s. t.

$$\sum_{j \in N_i} t_{ji}(\tau_{ji} e_{rx} + \tau^a_{ji} e^a_{tx}) + \sum_{j \in N_i} \sum_{k \in N_j, k \neq i} t_{jk} \tau_{jk} e^h_{rx} + \sum_{j \in N_i} t_{ij}(\tau_{ij} e_{tx} + \tau^a_{ij} e^a_{rx}) \leq E_i, \qquad \forall i \in N \backslash \{s\} \tag{5.11}$$

$$\sum_{j \in N_i} t_{ij} = \sum_{j \in N_i} t_{ji} + T, \qquad \forall i \in N \backslash \{s\} \tag{5.12}$$

$$t_{s,s} = T(|N| - 1), \tag{5.13}$$

with

$$\tau_{ij} = \begin{cases} \infty & prr_{ij} = 0 \vee prr_{ji} = 0, \\ \frac{1}{prr_{ij} prr_{ji}} & \text{otherwise}, \end{cases} \tag{5.14}$$

and $\tau^a_{ij} = prr_{ij} \tau_{ij}$.

Comparing LP 1 with LP 2 reveals that both differ significantly in their complexity. While LP 1 requires binary variables in Equations 5.5 and 5.6 to prohibit routing cycles, LP 2 needs only Equation 5.13 instead. But as long as t_{ij} is defined as an integer variable, both LPs will still have a high computational complexity because special algorithms are required to solve the integer constraints. Neglecting these constraints would lead to a heuristic solution. While LP 1 would then become a *mixed-integer program* (MIP) due to still required binary variables, LP 2 could be solved by standard linear programming using a common LP solver. Its complexity is orders of magnitude lower such that even larger networks can be solved in an acceptable time.

In order to find an approximation of LP 1, even for high node densities, LP 2 can be used as a starting point. Finding the optimal lifetime T^* of the network can then be performed by means of simulations. t^*_{ij} specifies for each link (i, j) the maximum number of packets which may be sent over that link. As the number of packets pertains to the entire lifetime, we use the following heuristic to determine when and how often a node i will forward packets over a link (i, j):

At first, we compute the fraction t'_{ij}, with which a neighbor j is used as a forwarder by a node i. According to the LP solution, t'_{ij} can be calculated as

$$t'_{ij} = \frac{t^*_{ij}}{\sum_{k \in N_i} t^*_{ik}}. \tag{5.15}$$

Then, all neighbors j are ordered in a list with $\alpha_i(j)$ being the index of j in the list and $\hat{\alpha}_i(k)$ being the node at index k. Since $\sum_{j \in N_i} t'_{ij} = 1$, the forwarding decision can be made randomly for each

packet that needs to be forwarded by node i. The decision is based on evaluating

$$\sum_{k=1}^{\alpha_i(j)-1} t'_{i,\hat{\alpha}_i(k)} < X \leq \sum_{k=1}^{\alpha_i(j)} t'_{i,\hat{\alpha}_i(k)}, \tag{5.16}$$

where X is a uniformly distributed random variable. The neighbor that fulfills constraint 5.16 is then selected as the forwarder. Over the lifetime of the network, the average load on each link (i, j) should thus be equal to t^*_{ij}[1]. In so doing, the solution obtained from the LP solver can be used to create the forwarding graph of the network[2].

Note that LP 1, as well as LP 2, requires global knowledge in order to maximize the network lifetime. Although distributed algorithms can be found, they are quite complex and require a significant overhead in order to exchange the required information among nodes. As we are more interested in the results obtained by linear programming, we have not implemented such distributed solutions and compute the maximum network lifetime in a centralized fashion. This may provide an interesting comparison with the performance of other approaches we consider in the following.

5.4 Lifetime-Efficient Forwarding

As stated at the end of Chapter 4, the energy-efficient forwarding strategies proposed may have the disadvantage that some paths are used more frequently than others; i. e., the majority of nodes along these paths will consume more energy, which will decrease their lifetime significantly. Thus, maximizing the network lifetime may be an orthogonal problem that needs to be taken into account in addition to the energy efficiency of the network. However, the optimal solution will depend on the definition of lifetime and on the performance aims which are supposed to be optimized. For example, if the network lifetime is defined as the time that must elapse before all nodes in the network are "dead", the lifetime is maximized by a *minimum energy consumption* strategy, without considering the end-to-end packet delivery ratio or the energy efficiency of forwarding paths. On the other hand, if the lifetime is defined as that of the first node to run out of energy, a forwarding strategy aiming to *maximize* the *minimum residual energy* on a path would be optimal. But as stated above, limiting the focus to the network lifetime may lead to worse packet delivery ratios.

Hence, the idea is to combine energy-efficient forwarding with maximum lifetime forwarding in order to find a good trade-off between both performance aims. With E_i^{eff} being the best energy efficiency of the forwarding path of a node i, each node tries to optimize

$$E_i^{eff} \cdot E_i^l \rightarrow \max. \tag{5.17}$$

E_i^l denotes the *expected* lifetime of the entire forwarding path towards the sink, which is influenced by the minimum residual energy along that path. Using the expected lifetime rather than the minimum

[1] Note that concerning the lifetime as well as the nodes' energy consumption it makes no difference *when* a packet is sent over a link, as long as not more than t^*_{ij} packets are sent and the packet reception ratios do not change over time.

[2] As the forwarding paths change over time due to the random process, a forwarding graph is established rather than a forwarding tree

lifetime accounts for the lifetime of forwarding paths much better. As the worst node (regarding its residual energy) along the path consumes energy for forwarding only if packets do not get lost enroute to it, the expected lifetime is more opportunistic and better estimates the real path cost. According to SEEF and MEEF as presented in Chapter 4, we call these strategies *single-link* and *multi-link lifetime-efficient forwarding* (SLEF and MLEF), respectively.

5.4.1 Analysis of the Finite Retransmissions Case

In order to calculate the expected lifetime of a forwarding path, we consider the case of finite retransmissions, for single-link as well as for multi-link lifetime-efficient forwarding[3]. The mathematical derivations are quite similar to the ones presented in Section 4.5 of Chapter 4 and rely on the same probability trees.

Single-Link Lifetime-Efficient Forwarding

Starting with the less complex case of single-link forwarding, Figure 5.1 shows the forwarding probability tree of a node i concerning the calculation of its expected path lifetime. As long as a packet is not successfully received by a considered forwarder j ($\bar{F}$), the path lifetime will be determined by the lifetimes of node i and j only, denoted as $L = \min\{l_i, l_j\}$. The node lifetimes l_i and l_j are defined as the residual energy of node i and j, respectively. Note that even if the transmission is not successful, it can be assumed that the receiver will spend energy trying to decode the packet. Thus, its residual energy must be taken into account. On the other hand, if the packet has been received by j correctly (F), the path lifetime will change to $\min\{L, E_j^l\}$, as j will try to forward the packet in any case. Its expected lifetime E_j^l thus influences the lifetime of node i, regardless of whether acknowledgements get lost and packets need to be retransmitted.

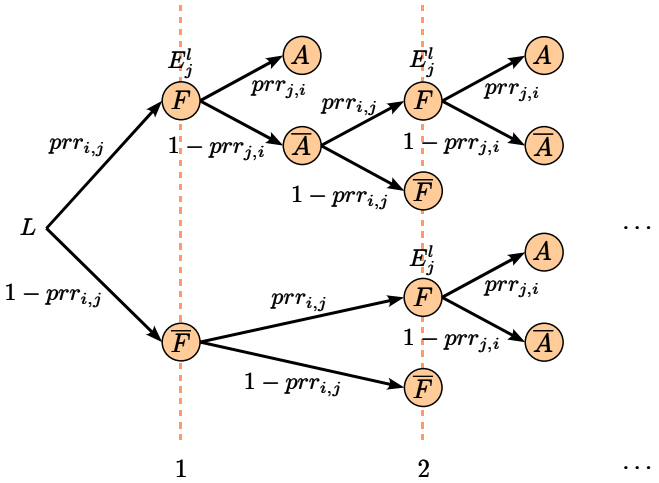


Figure 5.1: Probability tree for the lifetime of SLEF

[3]The infinite retransmissions case can afterwards be obtained from $R \to \infty$.

Thus, E_i^l is calculated as follows: Let $\hat{E}_l^k$ be a measure of the expected minimum residual node's energy on a forwarding path towards the sink if up to $k-1$ retransmissions are used. Then, $\hat{E}_l^{R+1}$ is defined iteratively as

$$\begin{aligned} \hat{E}_l^{R+1} &= prr_{ij} \min\{L, E_j^l\} + (1 - prr_{ij})\hat{E}_l^R \\ \hat{E}_l^R &= prr_{ij} \min\{L, E_j^l\} + (1 - prr_{ij})\hat{E}_l^{R-1} \\ &\vdots \\ \hat{E}_1 &= prr_{ij} \min\{L, E_j^l\} + (1 - prr_{ij})L. \end{aligned} \tag{5.18}$$

With E_i^l being equal to $\hat{E}_l^{R+1}$, the expected lifetime of node i is then

$$E_i^l = \left(1 - (1 - prr_{ij})^{R+1}\right) \min\{L, E_j^l\} + (1 - prr_{ij})^{R+1} L \tag{5.19}$$

for the single-link case. In other words, E_i^l refers to the expectation of the minimum residual energy of nodes that are affected if i tries to forward a packet to a neighbor j, which in turn may forward it towards the sink.

Multi-Link Lifetime-Efficient Forwarding

The calculation for the multi-link case is a little bit more complicated because we must take into account that packets may be forwarded by more than one node at the same time. Figure 5.2 illustrates the probability tree of MLEF.

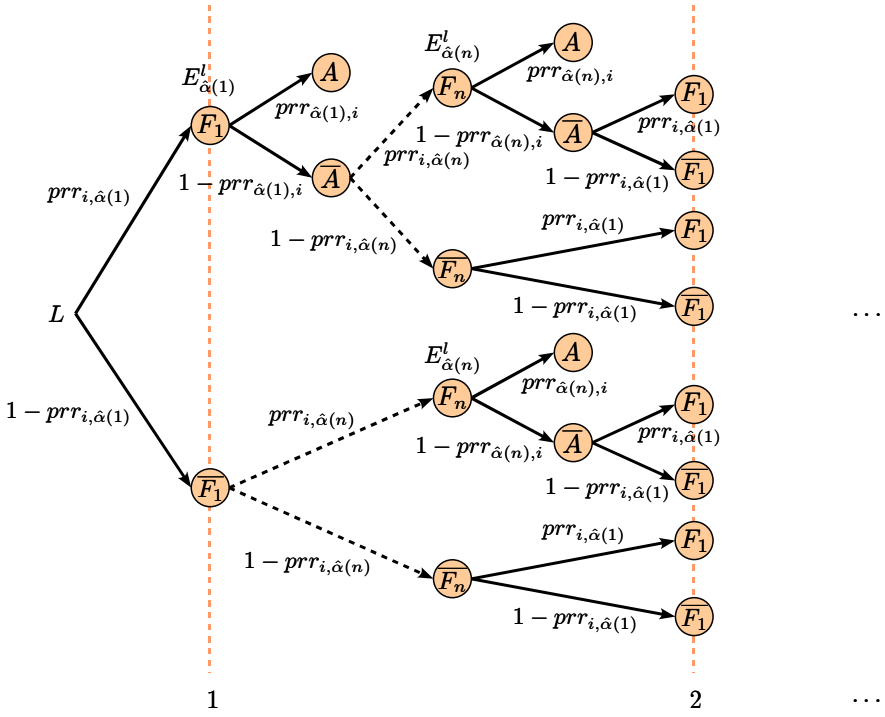


Figure 5.2: Probability tree for the lifetime of MLEF

If a packet cannot be forwarded successfully ($\overline{F_1} \wedge \ldots \wedge \overline{F_n}$), the expected lifetime is equal to $\min\{l_i, l_{\hat{\alpha}(1)} \ldots l_{\hat{\alpha}(n)}\}$. However, once one or more nodes start forwarding the packet, their appropriate path lifetimes E_j^l must be considered[4]. Hence, for $L = \min_{j \in \Omega_i}\{l_i, l_j\}$, the expected lifetime E_i^l is defined iteratively as $\hat{E}_l^{R+1}$ as follows:

$$
\begin{aligned}
\hat{E}_l^{R+1} &= prr_{i,\hat{\alpha}(1)} \Big[prr_{\hat{\alpha}(1),i} \min\{L, E^l_{\hat{\alpha}(1)}\} \\
&\quad + (1 - prr_{\hat{\alpha}(1),i}) \Big[prr_{i,\hat{\alpha}(2)} \Big[prr_{\hat{\alpha}(2),i} \min\{L, E^l_{\hat{\alpha}(1)}, \hat{E}^l_{\hat{\alpha}(2)}\} \\
&\quad\quad + (1 - prr_{\hat{\alpha}(2),i}) [\ldots]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(2)}) [\ldots]\Big]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(1)}) \Big[prr_{i,\hat{\alpha}(2)} \Big[prr_{\hat{\alpha}(2),i} \min\{L, E^l_{\hat{\alpha}(2)}\} \\
&\quad\quad + (1 - prr_{\hat{\alpha}(2),i}) \Big[\ldots + (1 - prr_{i,\hat{\alpha}(n)}) \min\{L, E^l_{\hat{\alpha}(2)}, \hat{E}^R_l\}\Big]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(2)}) \Big[\ldots + (1 - prr_{i,\hat{\alpha}(n)}) \min\{L, \hat{E}^R_l\}\Big]\Big] \\
&\vdots \\
\hat{E}_l^{1} &= prr_{i,\hat{\alpha}(1)} \Big[prr_{\hat{\alpha}(1),i} \min\{L, E^l_{\hat{\alpha}(1)}\} \\
&\quad + (1 - prr_{\hat{\alpha}(1),i}) \Big[prr_{i,\hat{\alpha}(2)} \Big[prr_{\hat{\alpha}(2),i} \min\{L, E^l_{\hat{\alpha}(1)}, \hat{E}^l_{\hat{\alpha}(2)}\} \\
&\quad\quad + (1 - prr_{\hat{\alpha}(2),i}) [\ldots]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(2)}) [\ldots]\Big]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(1)}) \Big[prr_{i,\hat{\alpha}(2)} \Big[prr_{\hat{\alpha}(2),i} \min\{L, E^l_{\hat{\alpha}(2)}\} \\
&\quad\quad + (1 - prr_{\hat{\alpha}(2),i}) \Big[\ldots + (1 - prr_{i,\hat{\alpha}(n)}) \min\{L, E^l_{\hat{\alpha}(2)}\}\Big]\Big] \\
&\quad + (1 - prr_{i,\hat{\alpha}(2)}) \big[\ldots + (1 - prr_{i,\hat{\alpha}(n)}) L\big]\big] .
\end{aligned}
\tag{5.20}
$$

Unfortunately, $\hat{E}_l^{R+1}$ cannot be expressed in a closed form for $n > 1$ due to the $\min$-operation. Thus, an exact computation can only be achieved by applying the iterative calculation directly.

Otherwise, the following approximations could be used: A pessimistic approximation would be to assume that packets will be forwarded by all nodes contained in Ω_i. In this case, E_i^l can be calculated from $\min\{L, E^l_{\hat{\alpha}(1)} \ldots E^l_{\hat{\alpha}(n)}\}$. On the other hand, assuming $prr_{j,i} = 1, \forall j \in \Omega_i$ would lead to an optimistic approximation. The equations in 5.20 can then be simplified to

$$
\begin{aligned}
\hat{E}_l^{R+1} &= \sum_{j \in \Omega_i} a_{i,\alpha(j)-1} prr_{i,j} \min\{L, E_j^l\} + a_{i,n} \hat{E}_l^R \\
&\vdots \\
\hat{E}_1 &= \sum_{j \in \Omega_i} a_{i,\alpha(j)-1} prr_{i,j} \min\{L, E_j^l\} + a_{i,n} L,
\end{aligned}
\tag{5.21}
$$

[4]Note that in case of lost acknowledgements, multiple forwarders may be polled although a packet has already been forwarded.

with $a_{i,k} = \prod_{j \in \Omega_i, \alpha(j) \leq k}(1 - prr_{i,j})$. E_i^l is then expressed as

$$E_i^l = \frac{\left(\sum_{j \in \Omega_i} a_{i,\alpha(j)-1} prr_{ij} \min\{L, E_j^l\}\right)\left(1 - a_{i,n}^{R+1}\right)}{1 - a_{i,n}} + a_{i,n}^{R+1} L. \tag{5.22}$$

However, for the simulations, as well as for the experimental evaluation, we use the exact calculations shown in 5.20. To limit the computational complexity, we set the maximum number of potential forwarders, i. e., the size of the forwarder set Ω, and the maximum number of retransmissions to three.

5.5 Evaluating the Forwarding Strategies

First, we will compare SLEF and MLEF with the non-lifetime-efficient forwarding strategies presented in Chapter 4 such as SEEF and MEEF, MT and MT2 forwarding, PRR-based forwarding, E_r-based forwarding, and hop-based forwarding. Afterwards, we will investigate the impact of a lifetime component which incorporates the residual energy levels of forwarding nodes. While SLEF and MLEF represent the lifetime extensions of SEEF and MEEF, the other strategies are extended as follows. Let $L = \min_{(j,*) \in \phi}\{l_j\}$ be the minimum residual energy on a forwarding path ϕ consisting of the set of forwarding links towards the sink.

- **MT-L Forwarding:** In contrast to MT forwarding, which minimizes the number of expected transmissions on a forwarding path, MT-L forwarding additionally downgrades paths with low-energy nodes by using the extended forwarding metric $\frac{1}{L}\sum_{(i,j) \in \phi} \frac{1}{prr_{i,j} prr_{j,i}} \rightarrow \min$.
- **PRR-L-based Forwarding:** Similarly, PRR-based forwarding is extended such that each node i selects the neighbor j that maximizes $\frac{|\phi|}{prr_{i,j} prr_{j,i}} \cdot L$ as its forwarder.
- **Hop-L-based Forwarding:** Based on a neighbors' hop counter denoted as $|\phi|$, the node which minimizes $\frac{|\phi|+1}{L}$ becomes the forwarder. As in hop-based forwarding, the node with the best reception rate will be selected in the case of equal values.
- **E_r-L-based Forwarding:** With $E_i^r = \prod_{(j,k) \in \phi} 1-(1-prr_{j,k})^{R+1}$ being the end-to-end packet delivery ratio of a node i, the neighbor that maximizes $E_i^r \cdot L$ becomes the forwarder of i. In case of equal values, the node with the smallest hop counter will be selected.
- **LP Relaxation:** In addition, we compare all forwarding strategies with the optimal lifetime solution obtained by linear programming. Due to the computational complexity, we only use an LP relaxation: The exact linear program for the finite retransmissions case is approximated by LP 2 as described in Section 5.3.2. That is, although nodes use a finite number of retransmissions during the data forwarding process, the forwarding paths are computed based on the assumption of an infinite number of retransmissions[5]; the LP 2 solution can be considered to be a relaxation of the optimal LP 1 solution.

[5] Note that even if infinite retransmissions are allowed, the expected number of retransmissions is finite and much lower.

5.6 Simulations

Similar to the simulations carried out in Chapter 4, let us consider a stationary network without any mobility where nodes are uniformly distributed over a 200 $\times$ 200 m^2 field. The nodes' maximum communication range is set to 30 m according to the packet reception model presented in Section 4.3 of Chapter 4, which is used to determine the probability of packet loss between two adjacent nodes in the network. The number of retransmissions is set to three. Again, we assume that the loss rate does not change with time and that each node will know the estimated link qualities by overhearing the wireless channel and counting packet losses, e. g., by using an exponential weighting function.

In order to calculate the overall network lifetime, the time is subdivided into rounds. The network lifetime is then defined as the number of rounds that have expired when the first node runs out of energy. Energy is consumed according to the first order model, which was also used by the EEF simulations carried out in Chapter 4. However, this time, each node has only an initial energy of 0.1 J, determining its lifetime.

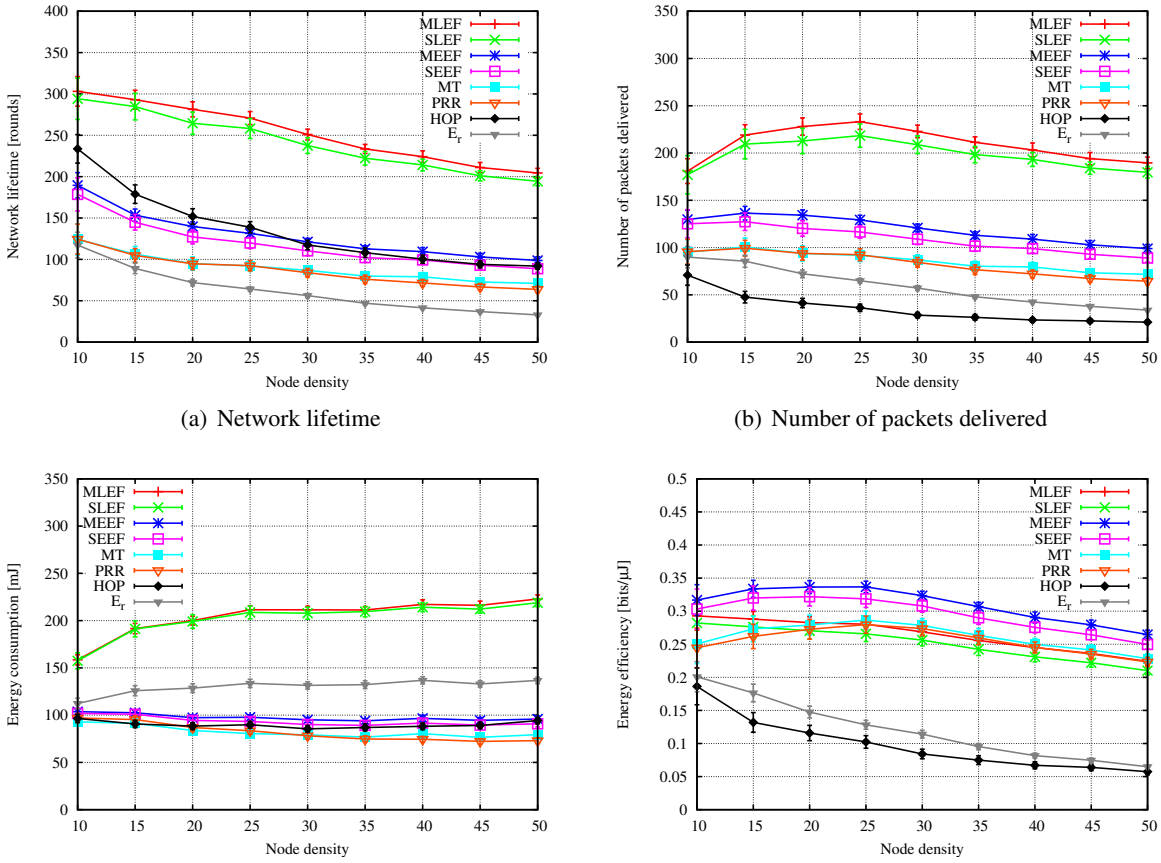
During each round, a certain number of nodes issues a data packet of 32 bytes that is then transferred along established forwarding paths towards a predefined network sink. The propagation time on a link and the processing time of the nodes are again neglected. With each round, the forwarding paths in the network may change due to decreasing energy levels, depending on the forwarding strategy being simulated. Thus, all nodes need to exchange their forwarding information periodically and to recompute forwarding paths and metrics if necessary.

All forwarding strategies described above are simulated for different node densities and fractions of source nodes issuing data packets. Simulations are carried out until the first node has consumed its entire energy. They are repeated 200 times; thus, the data points in the following graphs are averaged over 200 simulation runs and shown with their 0.95 quantiles.

5.6.1 Performance Comparison of LEF and EEF

While LEF aims to maximize lifetime efficiency, i. e., to prolong the network lifetime as well as to maximize the energy efficiency, it is surely interesting to note how it performs against plain energy-efficient forwarding, as considered in the previous chapter. Besides EEF, we also show the performance of MT, PRR, HOP, and E_r-based forwarding, which like EEF do not consider the residual energy on forwarding paths. In contrast to Chapter 4, the simulation is carried out over the entire network lifetime, using a source fraction α of 0.2. The results are illustrated in Figure 5.3 for different node densities, showing the network lifetime in rounds, the number of packets delivered and energy consumed per source node, and the energy efficiency at the end of the network's lifetime.

As shown in Figure 5.3(a), LEF achieves a significantly higher network lifetime than does EEF, while multi-link forwarding performs better in both cases. Thus, the extended forwarding metric of LEF does indeed prolong the lifetime of the network. Without considering LEF, hop-based forwarding and EEF outperform MT, PRR, and E_r-based forwarding, which can be attributed to their lower energy consumption per round, as we have already discussed in Chapter 4. However, in contrast to EEF, hop-



(a) Network lifetime

(b) Number of packets delivered

(c) Energy consumption

(d) Energy efficiency

Figure 5.3: Performance comparison of LEF and EEF ($\alpha = 0.2$, $R = 3$)

based forwarding suffers heavily from low delivery ratios per round and from poor energy efficiency. Despite its long network lifetime, the number of delivered data packets averaged over all source nodes is lower than for every other strategy, as illustrated in Figure 5.3(b). On the other hand, both MEEF and SEEF perform much better, even if their network lifetime is slightly shorter. Since MLEF, as well as SLEF, further extends the lifetime, more packets can be delivered until the first node runs out of energy. But due to a longer lifetime, LEF also consumes the most energy, followed by E_r-based forwarding, SEEF, and MEEF, as shown in Figure 5.3(c). E_r-based forwarding causes the highest energy consumption per round due to long forwarding paths in terms of hops. Comparing LEF and EEF concerning the energy consumed per round shows that LEF is outperformed by EEF substantially. Thus, avoiding low-energy nodes along forwarding paths, as done in LEF, causes a higher overall energy consumption but at the same time spreads the consumption among nodes better. This clearly identifies the trade-off between energy efficiency and network lifetime. While LEF improves the lifetime of EEF, it needs more energy in order to protect low-energy nodes, at the expense of poorer energy efficiency as shown in Figure 5.3(d).

Hence, only EEF achieves the sole maximization of energy efficiency. However, if issues pertaining to lifetime need to be taken into account by an application, LEF trades off energy efficiency and network lifetime much better. We account for this trade-off by the term *lifetime-efficient*, indicating the aim of maximizing lifetime as well as energy efficiency simultaneously. Figure 5.4 depicts the lifetime

efficiency defined as the product of network lifetime and energy efficiency for LEF and EEF, and in addition for MT, hop-based, PRR-based and E_r-based forwarding.

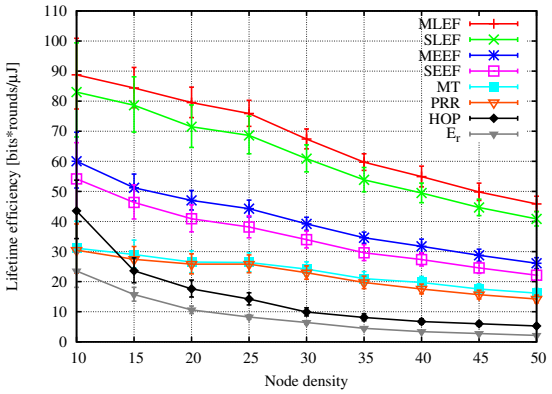


Figure 5.4: Lifetime efficiency comparing LEF and EEF ($\alpha = 0.2$, $R = 3$)

Both LEF strategies clearly outperform EEF due to a longer network lifetime and sufficient energy efficiency. As multi-link forwarding usually achieves a better energy efficiency, it improves the lifetime efficiency of single-link forwarding, too. However, without considering the residual energy of nodes, the performance of EEF is still superior to that of the remaining forwarding strategies. Thus, in conclusion, it may heavily depend on the application scenario whether or not lifetime-efficient or energy-efficient forwarding is preferable, favoring either LEF or EEF.

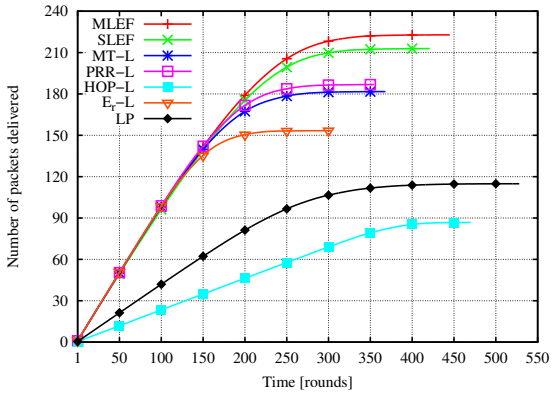
5.6.2 Network Performance over Time

In the following, we will investigate LEF regarding different performance issues in more detail, starting with its performance over time. We carried out simulations by using a density of 30 nodes, with 20% of all nodes acting as source nodes that generate one data packet per round. The simulations ran until the first node had consumed its energy completely.

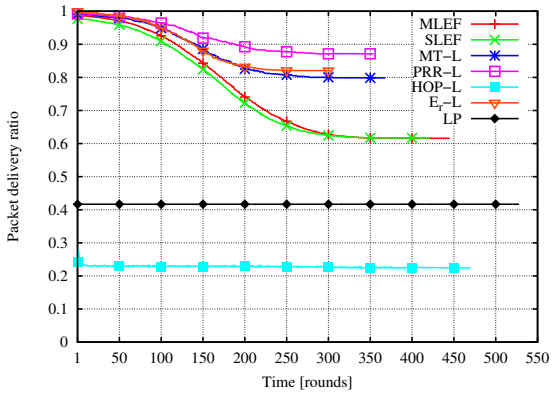
Figure 5.5 shows the simulation results for several performance metrics plotted over time. Additionally, it indicates the *maximum* network lifetime of each forwarding strategy obtained over all simulation runs[6]. We now consider the performance of each forwarding strategy separately, rather than describing each graph on its own.

Starting with hop-L-based forwarding, Figure 5.5(a) shows the average number of data packets delivered to the sink over time. Since blacklisting was not applied, many packets were sent over lossy links, causing significant packet losses. As the number of retransmissions is often insufficient in order to send packets over established long-distance forwarding links, packet drops occur quite often. For this reason, the number of delivered packets, as well as the packet delivery ratio, which is depicted in Figure 5.5(b), were worst compared to all other strategies. However, despite the fact that much energy was spent on retransmissions, energy might also be "saved" along a forwarding path once packets have been dropped, because forwarding then no longer takes place. This is illustrated in Figure 5.5(c),

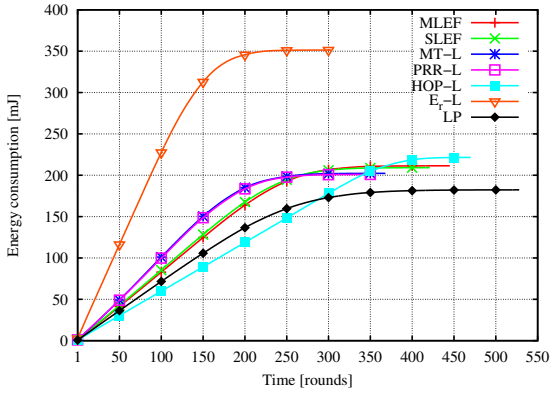
[6]Note that not the *average* but the *maximum* network lifetime is shown.



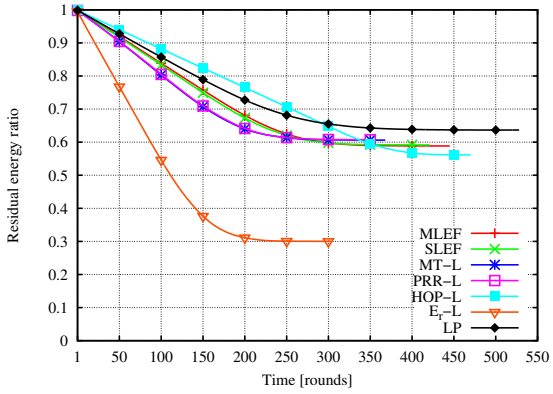
(a) Number of delivered packets



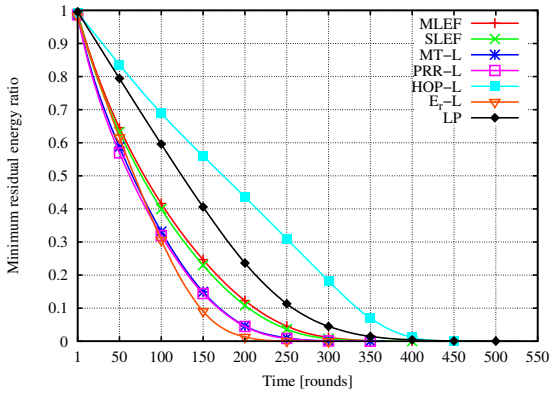
(b) Packet delivery ratio



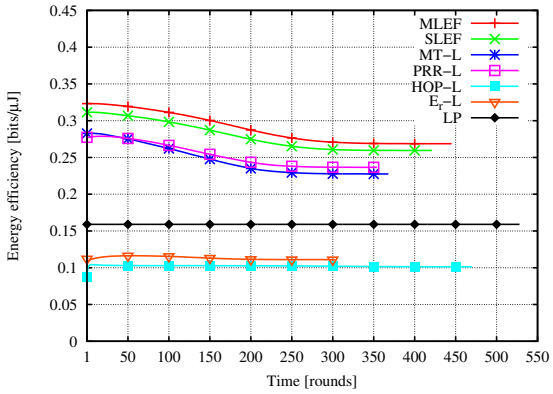
(c) Energy consumption



(d) Residual energy



(e) Minimum residual energy



(f) Energy efficiency

Figure 5.5: Forwarding performance over time ($\mu = 30$, $\alpha = 0.2$, $R = 3$)

which shows the overall energy consumption within the network over time. Since hop-L-based forwarding consumed the least amount of energy per round, the node's average as well as its minimum residual energy ratios were substantially higher, as shown in Figure 5.5(d) and 5.5(d). Thus, it was able to achieve quite a long network lifetime. However, at the same time, the network energy efficiency of hop-L-based forwarding was the worst, as a result of too many packet losses (see Figure 5.5(f)).

In contrast to hop-L-based forwarding, the LP solution does not "benefit" from packets that are dropped early on a forwarding path. Because an infinite number of retransmissions is assumed, bad forwarding links are often avoided since they cause a high energy consumption, which may shorten the network's lifetime. As a result, the number of delivered packets, and thus the packet delivery ratio at the end of the network's lifetime, are higher than for hop-L-based forwarding, as shown in Figure 5.5(a) and 5.5(b). However, compared to LEF, MT-L, PRR-L, and E_r-L-based forwarding, the difference is significant; only half the number of packets arrives at the sink. Because of this, less energy is consumed, which affects the node's average, as well as its the minimum, residual energy ratios "positively". But like for hop-L-based forwarding, Figure 5.5(f) depicts that the energy efficiency of the LP solution is quite poor and less than half that of LEF. Thus, the simulation results clearly show that maximum lifetime strategies might perform badly if packet delivery ratios are not taken into account. Especially if packet drops are exploited to extend the network's lifetime, will data forwarding become useless, since the most trivial solution of suppressing any packet transmission would then perform best.

On the other hand, considering solely the end-to-end delivery ratios as done in E_r-L-based forwarding does not lead to a long network lifetime and high energy efficiency either. Although at the beginning of the simulation E_r-L-based forwarding achieves the best delivery ratio, it starts suffering from a high energy consumption (see Figure 5.5(c) and 5.5(d)). Because forwarding paths are usually comparatively long in order to exploit short-distance links, many nodes are affected by forwarding, thus consuming much energy. Due to the rapidly decreasing amount of residual energy shown in Figure 5.5(d), other forwarding paths need to be selected. As a consequence, the packet delivery ratio drops, as depicted in Figure 5.5(b). Furthermore, the high energy consumption causes a fast decrease in residual energy, finally leading to the shortest network lifetime. Concerning the energy efficiency of E_r-L-based forwarding, the performance is thus only slightly better than that of hop-L-based forwarding. Hence, despite a higher delivery ratio, the extremely high energy consumption cannot be compensated.

MT-L and PRR-L-based forwarding perform quite similarly, although their forwarding metrics are completely different. However, both strategies take into account packet reception ratios as well as the forwarding path length. Concerning the number of delivered packets, respectively the delivery ratio per round (Figure 5.5(a) and 5.5(b)), PRR-L-based forwarding performs slightly better and outperforms all other strategies, but at the expense of a shorter lifetime. As Figure 5.5(c) shows, the energy consumption of both strategies is basically the same, as are the residual and the minimum residual energy ratios over time. Hence, due to a better delivery performance, PRR-L-based forwarding achieves an energy efficiency that slightly surpasses the efficiency of MT-L, but is still worse than that of SLEF and MLEF.

Although both SLEF and MLEF are outperformed by PRR-L-based forwarding with respect to their delivery ratios, as shown in Figure 5.5(b), both strategies benefit from longer network lifetimes. They are thus able to deliver more packets to the network sink (see Figure 5.5(a)). However, the delivery ratios decrease significantly over time. This is due to the fact that if the residual energy along forwarding paths decreases, it becomes difficult to maintain high quality paths. In such a case, the lifetime component of LEF starts to avoid low-energy nodes by establishing "bypass paths" instead, worsening the forwarding quality of such paths. Figure 5.5(f) shows this trade-off as the energy efficiency that

decreases with time. In this way, LEF differs from EEF. At the beginning of the simulation, LEF and EEF have the same energy efficiency. However, as soon as the residual energy of nodes decreases, the energy efficiency of LEF is affected adversely. While the efficiency of EEF does not degrade, the efficiency of LEF drops by about 15% until the end of its lifetime. But compared to all forwarding strategies considered in Figure 5.5, both SLEF and MLEF still demonstrate a superior performance. Again, multi-link forwarding is able to improve the efficiency and delivery ratio of SLEF. Moreover, as MLEF consumes less energy, it also improves the network lifetime.

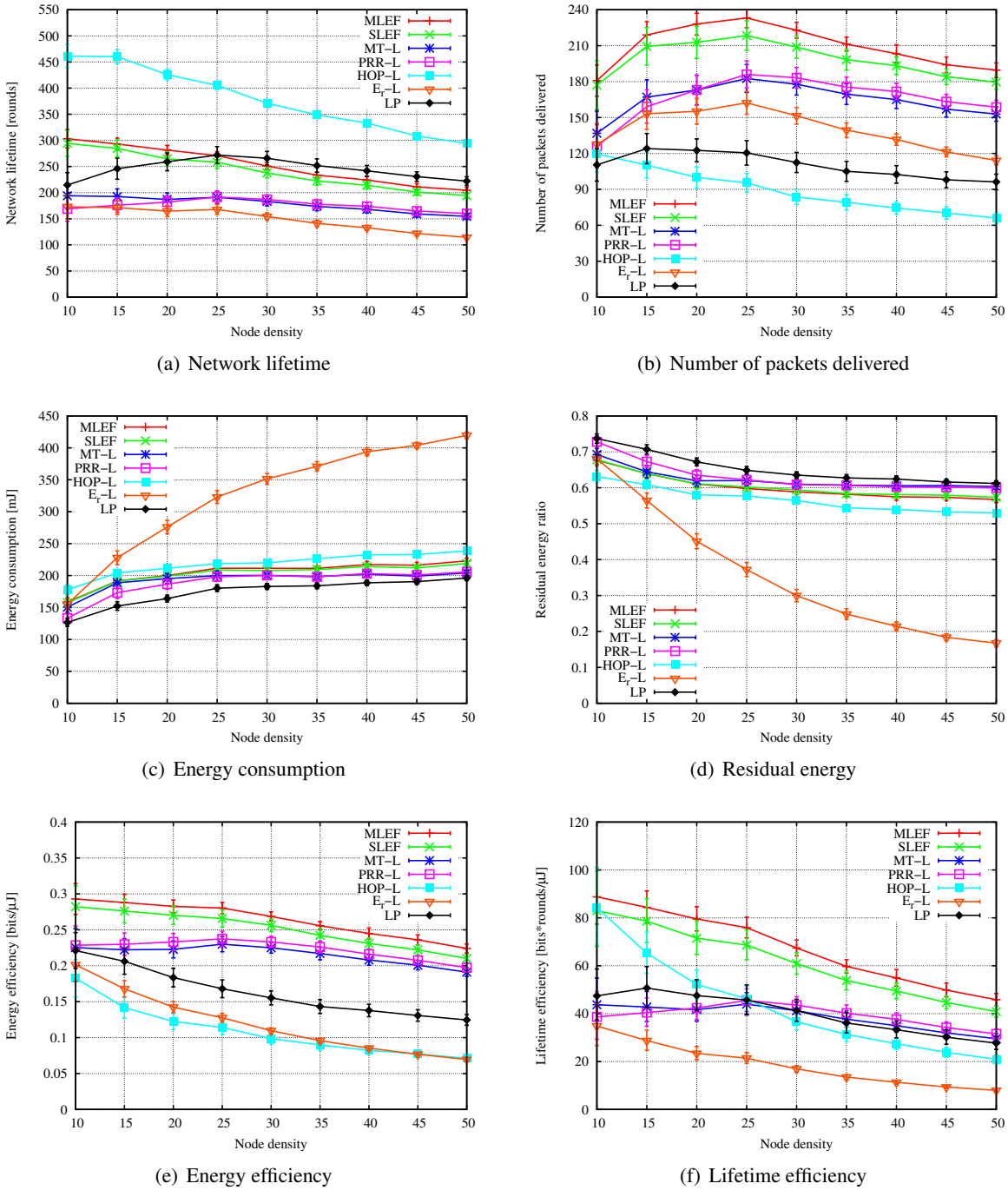
In conclusion, maximizing lifetime efficiency as done by both strategies trades off the contradictory aims of high delivery ratios, low energy consumption, and long network lifetimes very well. For example, although both the hop-L-based forwarding and the LP solutions achieve longer lifetimes due to less consumed energy, they suffer from bad delivery ratios and a low energy efficiency. In contrast, LEF spends more energy on packet forwarding, improving the delivery ratio and energy efficiency as well. Consequently, its network lifetime decreases. However, spending even more energy to further improve the delivery ratio, as done by E_r-based forwarding, is avoided for efficiency reasons.

5.6.3 Influence of Node Density

How the network performance is influenced by different node densities is discussed in this section. We use the same simulation setup as before, but additionally vary the node density between 10 and 50 nodes per maximum transmission range. Again, the fraction of source nodes is set to 0.2. Thus, at an increasing node density, the number of source nodes issuing data packets increases as well.

Figure 5.6 summarizes the performance characteristics of all forwarding strategies. In contrast to the maximum network lifetime that can be derived from Figure 5.5, Figure 5.6(a) depicts the *average* network lifetime, which might be quite different. For example, although the LP solution is optimized in terms of the lifetime of the network, it is outperformed by hop-L-based forwarding significantly. The reason is as follows: As discussed in the previous section, hop-L-based forwarding saves energy by sending packets over long-distance links, thus involving fewer nodes in the packet forwarding. As such links are commonly lossy, many packets will be dropped once the maximum number of retransmissions has been reached. Therefore, the resulting energy consumption per round will be significantly lower, which increases the network's lifetime. In contrast, the LP solution does not exploit this advantage because packet drops were not modeled by the LP relaxation. Since the network lifetime is optimized under the assumption of infinite retransmissions, the performance of the LP solution is worse than that of hop-L-based forwarding if the maximum number of retransmissions is limited to three, as in this simulation scenario.

In comparison, the lifetimes of MLEF and SLEF are shorter, but nevertheless longer than those of MT-L, PRR-L, and E_r-L-based forwarding. However, despite a shorter lifetime, both strategies are able to deliver more data packets to the sink, as shown in Figure 5.6(b). Since the lifetime of MLEF is a little bit longer and its energy consumption per round is smaller, it further improves the delivery ratio of SLEF, and outperforms all other strategies. Like in Figure 5.5, we see that although hop-L-based forwarding performs best regarding its lifetime, the total number of packets delivered over the entire lifetime is worst. The same applies for the LP solution. The initial increase can be explained as



(a) Network lifetime

(b) Number of packets delivered

(c) Energy consumption

(d) Residual energy

(e) Energy efficiency

(f) Lifetime efficiency

Figure 5.6: Influence of node density ($\alpha = 0.2$, $R = 3$)

follows: For a low node density, only a limited number of forwarding paths exists, which can be used to deliver packet successfully over a long time. If the node density increases, more path alternatives will become available and more nodes will issue data packets, which improves the number of packets delivered considerably. However, at the same time, the network's lifetime starts to decrease, as shown in Figure 5.6(a). Thus, for a very high node density, the delivery performance worsens again.

The total energy consumption over the entire lifetime is shown in Figure 5.6(c). In spite of shorter lifetimes, the energy consumption increased with an increasing node density, which is caused by the fact that number of source nodes increases, too. While almost all strategies are influenced similarly if more nodes are deployed, the energy consumption of E_r-L-based forwarding grows significantly. Because the energy consumption was not taken into account, the number of nodes affected by forwarding packets increases heavily in order to maximize the end-to-end delivery ratio. In addition, avoiding nodes which have little residual energy causes a even higher energy consumption. As a consequence, at a density of 50 nodes, the residual energy ratio at the end of the network lifetime was about 65% less than the residual energy of LEF (see Figure 5.6(d)).

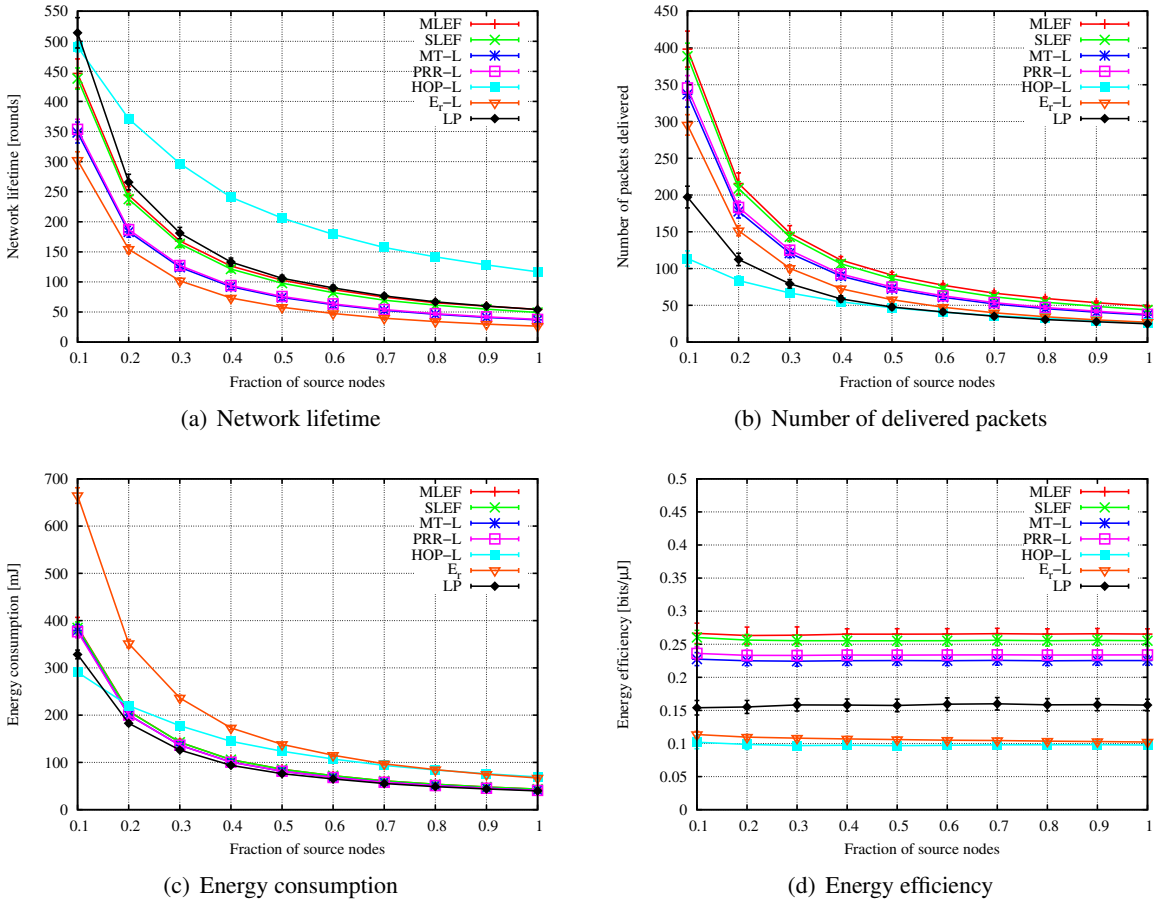
As depicted in Figure 5.6(e), the energy efficiency of the network tended to decrease if the number of deployed nodes started to increase. This is due to a higher energy consumption, a shorter network lifetime, and a lower number of delivered packets. Similar to Figure 5.5(f), MLEF and SLEF outperformed all other strategies clearly, independent of the density. Furthermore, both strategies achieved the best lifetime efficiency, as shown in Figure 5.6(f).

5.6.4 Influence of the Number of Source Nodes

Finally, we consider the influence of the number of source nodes on the network performance for a fixed density of 30 nodes per maximum transmission range. We increase the fraction of source nodes from 10% to 100%. The simulation results are shown in Figure 5.7. As illustrated in Figure 5.7(a), the network lifetime of all strategies drops about 80% to 90% if the source node fraction tends to one. As a result, the number of delivered packets, as well as the energy consumption over the entire lifetime, decreases (see Figure 5.7(b) and 5.7(c)). However, the relative performance among all strategies remains the same, independent of how many nodes issue data packets within the network. The fraction of source nodes influences the number of delivered packets, as well as the energy consumption, in the same way. The energy efficiency depicted in Figure 5.7(d) shows a constant behavior. This result is quite interesting because one might expect that the energy efficiency would be much better for a lower source fraction since less energy is consumed per round and the strategy's lifetime mechanism is less affected. However, as long as network congestion issues are not taken into account, the performance of all strategies remains stable. LEF again showed a superior performance in terms of energy as well as lifetime efficiency, even if the source node fraction changes.

5.7 Experimental Evaluation

In addition to the simulations, we also evaluated LEF and almost all other considered strategies by means of real-world experiments, using our WSN testbed consisting of 25 ESB nodes. Only the LP solution was not implemented because it is based on global knowledge and must be computed in a centralized way. Again, the evaluation results should be considered with respect to the small-sized network and the small number of ten evaluation runs. However, the obtained performance indications are quite sufficient for a first proof-of-concept.



(a) Network lifetime

(b) Number of delivered packets

(c) Energy consumption

(d) Energy efficiency

Figure 5.7: Influence of the number of source nodes ($\mu = 30$, $R = 3$)

5.7.1 Experimental Setup

The experimental setup is basically the same as described in Section 4.7.2 of Chapter 4. All nodes used a transmission power of 15% and retransmitted data packets up to three times if acknowledgements got lost. Data packets had a size of 32 bytes and were sent at a rate of five packets per minute, starting randomly in time. In order to avoid network congestion, only five source nodes were used that were randomly picked by the network sink. This represents a source fraction of 20%.

As before, we ran a pinging period before the experiment started in order to get estimates about packet reception ratios between pairs of network nodes; each node broadcast 100 ping packets in a round-robin fashion which were used to construct neighbor tables stored on each node individually. After that, the network sink triggered the establishment of forwarding tables by means of beacon packets that contained the appropriate forwarding metrics, as well as information about the estimated packet reception ratios to adjacent neighbors. In addition to the beacon content discussed in Section 4.7.2 of Chapter 4, each node also included its own or the minimum residual energy on its forwarding path, depending on the applied forwarding strategy. E. g., a node running LEF included the calculated E_i^l value, as well as its own energy level l_i.

In order to evaluate the network's lifetime and the lifetime efficiency of each forwarding strategy, each node used an initial energy of 100 mJ, which decreased with every received and transmitted packet according to the energy model presented in Section 4.3.3 of Chapter 4. The experimental evaluation stopped after the first node had consumed all its energy, determining the lifetime of the network and the end of the experiment. During the experiment, each node logged statistical information that was stored in the EEPROM of a ESB node and afterwards transmitted to the network sink.

As forwarding paths may change due to decreasing energy levels, all nodes continued broadcasting beacons at a rate of one beacon per minute. However, in case a forwarding path had changed, beacons were sent immediately after a short backoff time to inform adjacent neighbors. In doing so, frequently used nodes are prevented from running out of energy, as they propagated their decreasing energy levels to their neighborhood, forcing nearby nodes to search for alternative routes.

5.7.2 Evaluation Results

Each experiment was repeated ten times in order to minimize the influence of variations. Table 5.1 summarizes the main results, showing the average, as well as the 0.95 t-quantiles for different performance metrics. Additionally to LEF, MT-L, PRR-L, hop-L, and E_r-L-based forwarding, we also evaluated the performance of EEF to provide a comparison of network lifetime, energy efficiency, and lifetime efficiency. The structure of Table 5.1 is quite similar to that of Table 4.1 discussed in Chapter 4, except that now all values have been calculated with respect to the entire network lifetime.

The first row of Table 5.1 contains the network lifetime in minutes of each forwarding strategy. According to the simulation results, the experiments showed similar characteristics. LEF achieved a significantly longer lifetime than did EEF, which in contrast outperformed LEF in terms of energy efficiency. Furthermore, hop-L-based forwarding achieved the longest, and E_r-L-based forwarding the shortest lifetime. The performance of MT-L and PRR-L-based forwarding was almost the same and similar to EEF concerning the network's lifetime. Again, hop-L-based forwarding "benefited" from bad link qualities, long-distance links and a small network size, which is indicated by a lower number of transmitted acknowledgements (tx control), a low hop counter, and the high number of retransmissions. Mainly due to the small network delimiter, hop-L-based forwarding thus achieved a high lifetime efficiency, which was exceeded only by LEF. However, it was clearly outperformed concerning its energy efficiency. Only the efficiency of E_r-L-based forwarding was worse (it caused the highest energy consumption). Although the lifetime of E_r-L-based forwarding was less than half the lifetime of hop-L-based forwarding, it consumed about 15% more energy. As the simulations have already indicated, many nodes were involved in the forwarding process of E_r-L-based forwarding, which is shown by the number of packets received and transmitted, as well as by longer forwarding paths. Consequently, maximizing solely the end-to-end delivery ratio achieved the worst performance concerning the network's lifetime, lifetime efficiency, and energy efficiency.

Regarding the number of packets delivered over the entire lifetime, MLEF, as well as SLEF, showed the best results. If, in contrast, the number of packets delivered per minute is considered, MEEF and SEEF performed better. Moreover, as neither strategy caused any energy cost due to avoiding low-energy nodes, their network energy efficiency, as well as their energy efficiency per node, is about

Strategy	MLEF	SLEF	MEEF
Lifetime [min]	61.90 [52.82, 70.98]	59.40 [47.06, 71.74]	38.30 [33.99, 42.61]
Lifetime efficiency [bits·min/μJ]	30.58 [26.05, 35.10]	28.76 [24.03, 33.49]	19.68 [17.27, 22.09]
Energy efficiency [bits/μJ]	0.4940 [0.4182, 0.5697]	0.4842 [0.4263, 0.5431]	0.5241 [0.4542, 0.5942]
Energy efficiency per node [bits/μJ]	0.2782 [0.2498, 0.3067]	0.2728 [0.2511, 0.2946]	0.2994 [0.2693, 0.3300]
Tx data	328.50 [294.89, 362.10]	340.85 [305.18, 376.52]	215.15 [201.10, 229.21]
Tx control	104.54 [90.65, 118.43]	93.77 [79.43, 108.11]	90.48 [77.27, 103.70]
Rx data	149.93 [124.92, 174.93]	141.45 [121.82, 161.08]	96.82 [80.95, 112.69]
Rx control	172.78 [147.57, 198.00]	148.29 [121.98, 174.60]	139.85 [111.97, 167.72]
Packets delivered	2114.39 [1717.73, 2511.04]	1993.80 [1529.44, 2458.16]	1387.40 [1290.59, 1484.21]
Packets delivered per min.	34.15 [29.00, 39.30]	33.56 [27.14, 39.98]	36.22 [33.25, 39.19]
Energy consumption [mJ]	1081.84 [972.58, 1191.09]	1042.96 [951.51, 1134.42]	680.70 [620.00, 741.40]
Energy consumption per min. [mJ]	17.48 [15.61, 19.35]	17.56 [15.32, 19.80]	17.77 [16.22, 19.32]
Hop counter	2.51 [2.27, 2.76]	2.47 [2.23, 2.72]	2.84 [2.41, 3.27]
Retransmissions	1.13 [1.03, 1.23]	1.16 [1.02, 1.29]	0.85 [0.69, 1.00]

Strategy	SEEF	MT-L	PRR-L
Lifetime [min]	37.20 [34.55, 39.85]	40.60 [30.64, 50.56]	38.90 [31.06, 46.74]
Lifetime efficiency [bits·min/μJ]	19.13 [17.36, 20.90]	15.92 [12.77, 19.07]	15.67 [13.10, 18.24]
Energy efficiency [bits/μJ]	0.5144 [0.4539, 0.5749]	0.3923 [0.3213, 0.4633]	0.4027 [0.3485, 0.4569]
Energy efficiency per node [bits/μJ]	0.2919 [0.2652, 0.3185]	0.2429 [0.2082, 0.2776]	0.2476 [0.2228, 0.2725]
Tx data	222.82 [203.51, 242.14]	302.50 [251.82, 353.17]	285.73 [230.40, 341.05]
Tx control	74.18 [60.70, 87.65]	116.78 [88.89, 144.68]	118.96 [90.24, 147.68]
Rx data	84.28 [65.88, 102.68]	140.02 [116.81, 163.23]	138.40 [114.16, 162.63]
Rx control	110.38 [94.48, 126.28]	161.09 [119.08, 203.10]	161.00 [118.93, 203.07]
Packets delivered	1329.80 [1158.72, 1500.88]	1475.40 [996.52, 1954.28]	1414.00 [1025.68, 1802.32]
Packets delivered per min.	35.75 [32.63, 38.87]	36.34 [26.99, 45.69]	36.35 [28.82, 43.88]
Energy consumption [mJ]	663.24 [620.63, 705.86]	934.21 [774.04, 1094.37]	884.99 [711.09, 1058.90]
Energy consumption per min. [mJ]	17.83 [16.82, 18.84]	23.01 [18.77, 27.25]	22.75 [18.82, 26.68]
Hop counter	2.72 [2.31, 3.13]	2.93 [2.71, 3.15]	3.20 [2.94, 3.47]
Retransmissions	0.92 [0.73, 1.11]	0.94 [0.77, 1.10]	0.81 [0.66, 0.96]

Strategy	HOP-L	E_r-L
Lifetime	72.30 [62.70, 81.90]	35.10 [30.88, 39.32]
Lifetime efficiency [bits·min/μJ]	25.46 [19.72, 31.19]	9.26 [8.05, 10.47]
Energy efficiency [bits/μJ]	0.3521 [0.2406, 0.4636]	0.2637 [0.2252, 0.3021]
Energy efficiency per node [bits/μJ]	0.2035 [0.1592, 0.2477]	0.1799 [0.1624, 0.1975]
Tx data	380.74 [346.02, 415.46]	426.43 [399.62, 453.24]
Tx control	53.84 [49.08, 58.60]	182.30 [147.47, 217.13]
Rx data	123.85 [74.58, 173.13]	213.88 [183.20, 244.57]
Rx control	96.95 [82.11, 111.78]	242.47 [200.03, 284.91]
Packets delivered	1519.80 [1129.17, 1910.43]	1347.60 [1186.51, 1508.69]
Packets delivered per min.	21.02 [16.55, 25.49]	38.39 [34.15, 42.63]
Energy consumption [mJ]	1144.97 [1046.10, 1243.84]	1321.45 [1233.43, 1409.47]
Energy consumption per min. [mJ]	15.84 [14.74, 16.94]	37.65 [35.65, 39.65]
Hop counter	1.69 [1.52, 1.86]	3.53 [3.08, 3.98]
Retransmissions	1.92 [1.76, 2.09]	0.98 [0.81, 1.15]

Table 5.1: Results of the experimental evaluation (LEF)

6%-7% higher. However, due to a shorter network lifetime, the lifetime efficiency is about 33%-35% worse. Comparing the hop counters and the average number of retransmissions used in LEF and in EEF indicates that forwarding paths used by LEF become slightly shorter and lossier over time. Thus, the number of long-distance and direct links will increase in order to prevent intermediate nodes from running out of energy. For efficiency reasons, low-energy nodes will be by-passed by forwarding packets more directly towards the sink. Although this will cause more retransmissions, the energy consumption of forwarding nodes can be reduced. However, at the same time, the average energy consumption per link will grow, which will decrease the energy efficiency on the end-to-end forwarding path. But as nodes with low residual energy will be less affected, the network's lifetime will be extended, maximizing the lifetime efficiency of the network.

Comparing the multi-link and the single-link concepts used by MLEF and SLEF, respectively MEEF and SEEF, shows that multi-link forwarding was indeed applied within the network, improving the network lifetime as well as the energy and lifetime efficiency. Furthermore, using more than one receiver at the same time also reduced the number of data retransmissions. While single-link forwarding retransmitted data packets once the receiving node had not sent an acknowledgement back, multi-link forwarding exploited other nodes, which had received the packet successfully, more efficiently. As a result, the number of transmissions could be reduced significantly, which is shown by tx data in Table 5.1. However, multi-link forwarding caused more control packets (tx control) and more nodes to receive data packets (rx data). But as discussed above, these costs were compensated due to a better energy efficiency.

In conclusion, the experimental results confirm our simulations, highlighting LEF as the best strategy, maximizing both network lifetime and energy efficiency. Both evaluations clearly indicate that extending the network's lifetime comes at the expense of less efficiency. Thus, it is actually not possible to provide *the* optimal solution for general use. Rather, the application must define whether the primary objective is energy efficiency or lifetime efficiency.

5.8 Conclusions

In this chapter, we have provided an extension for energy-efficient forwarding that aims to maximize the lifetime efficiency of the entire network rather than the energy efficiency. By adding a lifetime component which takes the residual energy on forwarding paths into account, LEF is able to tackle the trade-off between long network lifetimes and an efficient usage of energy very well. Similar to EEF, we have proposed two versions, called SLEF and MLEF, to also account for multi-link forwarding. Both strategies have been theoretically analyzed, evaluated by means of simulations, and tested with real-world experiments. The obtained results have shown significant performance gains over EEF and other forwarding strategies.

In particular, we pointed out that solely maximizing the network lifetime might have bad performance characteristics in terms of the number of delivered packets, the energy consumption, and the energy efficiency. We presented a linear program which can be used by a common LP solver to solve the maximization problem. However, taking routing cycles into account requires a great computational

effort if packet drops are modeled. We thus relaxed the actual problem by the assumption of an infinite number of retransmissions. This relaxation is actually quite reasonable because otherwise packet drops might be heavily exploited due to energy savings. Nevertheless, the LP solution performed substantially worse, as the number of actually delivered packets was not taken into account.

Because LEF aims to maximize lifetime efficiency, it is outperformed by EEF concerning energy efficiency. Avoiding low-energy nodes thus comes at the expense of efficiency, likely increasing the overall energy consumption of the network. In future work, the efficiency of LEF might thus be improved by not giving all nodes equal consideration. For example, assume a two-tiered network consisting of few, but more important, sensing nodes and several redundant, and less important, forwarding nodes. Due to a higher number of forwarding nodes, an application task should not be affected if some of these nodes died. Thus, as long as a node finds enough neighbors in its vicinity, its residual energy level will be less important and need not be taken into account. That is, a forwarding node i propagates a modified energy value $\hat{l}_i$ that is calculated as

$$\hat{l}_i = \begin{cases} l_i & l_i < \xi \text{ or } N_i < \eta, \\ 1 & \text{otherwise,} \end{cases}$$

with ξ being a predefined energy threshold, N_i being the current number of active neighbors of node i, and η being a density threshold. Because nodes capable of sensing are considered to be more important, they will always propagate their real residual energy level, i. e., in this case $\hat{l}_i$ is equal to l_i. In this way, the advantages of LEF and EEF can be exploited much better. Only sensors and forwarding nodes having few neighbors are protected from being used too frequently by forwarding packets. Thus, the energy efficiency of the majority of forwarding paths could be optimized without much degrading the lifetime of the network.

Another idea for saving energy and thus extending the network's lifetime relies on exploiting the data content itself. Often data is issued by sensing nodes due to a physical phenomenon, which is likely to be sensed also by other nodes located in the vicinity. The correlation between these data readings might be exploited during the forwarding process by in-network processing. For example, if several data values arrive at an intermediate node along the forwarding path, aggregating the data might considerably reduce the number of packets to be forwarded. Forwarding data would thus require less energy, thereby increasing the network's lifetime, as well as the energy efficiency of the entire network. Furthermore, there might even be several cases where an application is only interested in aggregated information, e. g., in the average temperature or humidity in a habitat over a certain period of time.

How the possibility of data aggregation affects EEF and how the energy efficiency can be improved in such a scenario is considered in the following chapter.

CHAPTER 6

Energy-Efficient Aggregation Forwarding

"It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free."

– S. McConnell –

6.1 Introduction

The last two chapters have considered the energy efficiency and the lifetime efficiency of wireless sensor networks and focused on the network's delivery performance, as well as its energy consumption. As the radio transceiver is one of the major energy consumers, and it is expected that most of the sensor nodes will carry only a limited and irreplaceable power supply, such energy-efficient algorithms will be essential to provide a reliable infrastructure for any kind of application.

We have seen that the network's lifetime can be extended in many ways, and several protocols have been proposed in the last few years, affecting different layers in the protocol stack. Taking advantage of *data aggregation* is part of this chapter. While energy savings due to topology management are analyzed in the next chapter, we focus here on data aggregation in the context of energy-efficient forwarding, i. e., in which data received from adjacent neighbors is first *combined*, or *aggregated*, before it is sent over an energy-efficient forwarding path. Exploiting that kind of aggregation may reduce the communication overhead significantly, especially if it has already been taken into account during the establishment of forwarding paths. Since less energy will be consumed if packets are sent over nodes that exploit aggregation, the energy efficiency of the network will increase. However, as we will see, the cost reduction due to aggregation can be considerably higher if the forwarding strategy is aware of it.

The potential of data aggregation can be motivated as follows: In typical applications like habitat and environmental monitoring, disaster detection, or military surveillance applications, the detection of an

event or a particular *stimulus* will be in general first processed by a node and then forwarded to one or several sink nodes through the network. As it is likely that several nodes will report the same event, information received from different nodes may be combined, such that only one packet will need to be forwarded towards the sink. Particularly if the source nodes are close together, the correlation of data readings will probably be very high. Thus, aggregating these readings would reduce the packet redundancy significantly. In addition to such *event-based* aggregation, data queries issued by the sink are another example in which data aggregation might be useful if the sink is not interested in each reading individually. Examples of such *query-based* aggregation are, e. g., temperature or humidity queries.

By means of *in-network* processing, data gathering is done as follows: Based on the aggregation tree, which can be considered as a reverse multicast tree rooted at the sink, inner nodes will aggregate the readings from their child nodes before forwarding the data to their parent nodes upwards the tree. Aggregation functions are, for example, `SUM`, `COUNT`, `AVG` and `MIN/MAX` [164]. While these functions require little memory and low processing capabilities, they are limited to relatively simple types of queries; more advanced queries like approximations for the median, the most frequent data values, or data distribution histograms are also possible but require a higher overhead [222].

In contrast to classic *address-centric* routing, where packets are routed based on unique destination addresses and data are not changed, routing along aggregation trees is termed *data-centric* routing since inner nodes may perform any kind of in-network processing [112, 124, 135]. Although the idea of data-centric routing in conjunction with data aggregation was already proposed several years ago, it is still an open problem as to *how* aggregation trees should be constructed to be optimal [29, 74, 125, 145]. Regarding this issue, our contribution is an *energy-efficient aggregation forwarding scheme* that tries to find the best energy-efficient spanning tree that is optimized in terms of both the packet delivery ratio and energy cost under the capability of data aggregation.

The remainder of this chapter is structured as follows. In the next section, we first outline related work. Based on single-link and multi-link energy-efficient forwarding (see Chapter 4), which did not consider in-network processing explicitly, Section 6.3 presents an extension to SEEF and MEEF that accounts for the ability to aggregate. In addition to these *energy-efficient aggregation forwarding* strategies, Section 6.4 discusses two other possible approaches, which take advantage of global knowledge, but which are useful for comparison. Simulation results are presented in Section 6.5. Results obtained from real-world measurements are described in Section 6.6. Finally, Section 6.7 provides some concluding remarks.

6.2 Related Work

A first proposal for data-centric routing was *directed diffusion* [123, 124], which introduced a new communication paradigm that was so far unknown to address-centric routing: Initially, an *interest* is flooded throughout the network that is used to create reverse paths towards the node that is interested in the data. Then, data travel back along several paths established in the first phase. After receiving the first data packets, the sink reinforces the preferred paths in the network by sending *reinforcement*

packets to the source nodes. Those paths are used at a higher data rate, while the additional paths are maintained with lower data rates as backup paths. In [122], Intanagonwiwat *et al.* extended directed diffusion to facilitate data aggregation by means of greedy increment trees [234]. However, it performed poorly in large-scale sensor networks due to the global flooding. If geographic information is available, the performance can be improved, since then flooding can be restricted to particular regions. This is also proposed by TTDD [162, 267], which uses local flooding on a two-tier grid structure in order to facilitate large-scale data dissemination.

Similar to the directed diffusion approach, multi-path routing was proposed in several other approaches to improve reliability and account for packet losses. For example, instead of using spanning trees, directed acyclic graphs could be used. However, multi-path routing may cause message duplicates, which may lead to an overcounting of multiple readings and thus distort the aggregation results. Nath *et al.* have proposed a solution to this problem by means of *synopsis diffusion* [180], which provides a general framework to combine multi-path routing schemes with techniques to avoid double-counting. Yang *et al.* [264] have dealt with the problem of compromised nodes that may influence the trustworthiness of aggregation results, and proposed a secure aggregation protocol that minimizes the impact of compromised nodes on the aggregation accuracy by taking multiple aggregation functions into account. Westhoff *et al.* [251] studied the problem of end-to-end encryption in the presence of in-network processing, which is quite challenging if sensed data should be concealed end-to-end but also efficiently aggregated on the way to the the final destination. Therefore, a particular class of encryption transformations is required in order to aggregate values without the need for intermediate nodes to decrypt them.

Similar to the approach used by TTDD [162, 267], which divides the network into grids to perform data aggregation, Younis and Fahmy [272] propose HEED, a *hybrid, energy-efficient, distributed clustering approach*. Each cluster is assigned to a *cluster head* that aggregates all readings from nodes belonging to the cluster. The cluster heads are selected based on a combination of node degree and residual energy. In order to connect adjacent clusters, different transmission powers will be used.

Boukerche *et al.* [29] proposed an aggregation tree construction, in which leaf nodes will go into a low-power sleep mode in order to save energy. Non-leaves will stay awake and participate in data dissemination. The tree is constructed based on the node's residual energy and is optimized regarding the number of leaf nodes that will stay active. However, finding the optimum tree is NP-complete because the problem is equivalent to the minimum connected dominating set problem described in [93]. Boukerche *et al.* thus proposed an approximation that tries to optimize the spanning tree. However, the resultant overhead is quite significant.

A simpler algorithm is presented in [74], which uses the following idea: Upon receiving a message from a neighbor, a node will start a timer that is inversely proportional to its residual energy. While the timer is running and another message is received, the timer will be refreshed. If the timer expires, the node will become active. Thus, in the best case, the constructed spanning tree will be equal to the minimum spanning tree (MST), in which the costs of links are defined as the residual energy of the parent node.

In [143], Lee and Wong present E-Span, an *energy-aware spanning tree algorithm* that is rooted at the source node with the highest residual energy. Similar to [29] and [74], aggregation is performed by inner nodes of the tree, which are selected according to their residual energy and distance from the root (the selected source node) in terms of hops. After the aggregation tree has been established, the root will gather all data issued by other sources in the network and forward them to the actual network sink, again over the shortest hop path.

A similar approach that constructs a *lifetime-preserving tree* (LPT) is considered in [144]. In contrast to [143], the tree is rooted at the source node with the highest *tree energy*, which is defined as the minimum residual energy of all non-leaf nodes. Parent nodes will be selected such that the minimum residual energy on the path towards the root will be maximized. In so doing, LPT is able to achieve a better network lifetime than does E-Span [145]. However, the LPT algorithm is more complex and requires that each node propagate its forwarding path to its neighbors.

In [240], an approach for *minimum energy convergecast* is proposed that uses a heuristic solution in order to construct an aggregation tree with minimum energy costs, low latencies, and high reliability. It also proposes an algorithm for channel allocation by assigning nodes different transmission and reception codes if several communication channels are available. However, the tree construction works in a centralized way and requires global knowledge about connectivity.

Jia *et al.* [125] have proposed an interesting idea with GIST, a *group-independent spanning tree* that considers the case of unknown sources, which they called group-independent. Their aim is to find one single optimum spanning tree that can be used for all subsets of source nodes. The idea is to build the tree only once and then use it for any group of sources that issue data packets addressed to a sink. Such a group-independent spanning tree can be constructed by a hierarchical approach. By dividing the network into grids of equal size, GIST first establishes a so-called quad-tree. For each tree level, four adjacent grid cells are combined into a new cell, in which one *leader* is selected. The leader of the entire network will be the sink node, which will represent the root of the quad-tree. Using this tree structure, data will always be sent to the leader of the upper level, where it will likely be aggregated with data issued by other sources nodes. Since nodes at different levels are likely not within each others communication range, GIST is based on an underlying routing layer. The authors have shown that the cost of the induced aggregation subtree is within a logarithmic factor of the optimal solution for any kind of group. In the worst case, the performance of GIST will be better than that of MST or SPT, if both are constructed only once and then used for any combination of source nodes. However, a drawback of GIST is surely that each node must be aware of its location, which may not be possible at all times, even though several location systems already exist [32, 60, 119, 182]

Another idea that does not rely on forwarding trees is to use *gossip* algorithms [15, 30, 56], in which a message is forwarded to a random neighbor. By means of such gossiping schemes, the *averaging problem* can easily be solved, i. e., all nodes in the network are able to compute the average of all other nodes' measurements. In gossiping, each node randomly picks one of his one-hop neighbors, and both nodes exchange their current values. Afterwards, the average will become the new value for both nodes. By repeating this pairwise averaging, all nodes will finally converge to the global average. However, a common problem of gossiping is that information will diffuse throughout the network slowly if the randomization is restricted to the nodes' one-hop neighborhood. Dimakis *et*

al. [73] thus proposed geographic gossip, which reduces the energy consumption of standard gossip algorithms substantially. Rather than picking a random one-hop neighbor, each node picks a random location to which the node's current value will be forwarded, by means of geographic routing [170]. The nearest node to this location will then randomly decide whether or not the message is *accepted*. If so, the node will compute the average value and send its own value back to the original sender. Otherwise, a new random location will be picked. Although such gossip algorithms are quite robust, the energy efficiency is much worse than that of algorithms which explicitly establish aggregation trees, as proposed in this chapter.

In contrast to our work, most of the work proposed in the literature relies on opportunistic aggregation, i. e., the reduction of energy cost due to aggregation is not taken into account during the construction of a forwarding tree. Furthermore, packet losses due to poor link qualities are often neglected. However, taking advantage also of lossy links may improve the overall network performance. In the following, we thus present an energy-efficient algorithm which does take these issues into consideration. The aggregation tree functions in a distributed manner and improves the network's energy efficiency considerably, by exploiting aggregation gains *a priori*.

6.3 Energy-Efficient Aggregation Forwarding

Both of the energy-efficient forwarding strategies SEEF and MEEF described in Chapter 4 assume that data packets will be issued independently of each other. Furthermore, the information contained in different packets is assumed to be uncorrelated, which would prevent it from being aggregated. However, there are many cases in which the user of a sensor network will not be interested in each data reading of a node individually but rather in an aggregation value. For example, in a habitat environment, a user could be interested in the average, or the minimum and maximum, temperature values. In this case, several data readings may be aggregated into a single one along the forwarding path.

In terms of energy efficiency, data aggregation has a high impact, as the energy consumed to forward packets decreases. At the same time, the average packet delivery ratio, or to be more precise, the information delivery ratio, will not change, regardless of whether or not aggregation is performed within the forwarding tree[1], which increases the energy efficiency of the network.

6.3.1 Construction of the Aggregation Tree

Unfortunately, the problem of constructing an aggregation tree is equal to the Steiner tree problem, which is NP-hard [93]. Thus, in order to find the optimum aggregation tree, the problem of the *Steiner minimum tree* (SMT) must be solved. An SMT is a tree within a graph of N nodes, connecting n source nodes with minimum path costs. Generally, n is smaller than N since otherwise, the SMT will be equal to the MST that has polynomial costs. For $n < N$, it is very challenging to find the optimal aggregation nodes within the graph.

[1] Assuming that packets will get lost independently of each other.

We thus simplify the problem to the case in which only source nodes are assumed to aggregate data packets. All other nodes are assumed to purely forward data without any kind of aggregation. It should be noted that this simplification is just required to establish the aggregation tree. Afterwards, data may be aggregated at each node that has received data packets from more than one source.

The simplification allows us to construct an MST approximation on the overlay graph, which connects all source nodes with minimum costs. The path costs between two sources are defined according to the end-to-end energy efficiency metric, as used by SEEF and MEEF. While the end-to-end delivery ratio E_i^r of a node i will not change in comparison to that of EEF, the expected energy costs E_i^e will decrease if the information issued by node i can be aggregated along its forwarding path. In other words, if i is a source node, it will set E_i^e to zero, because neighbors forwarding packets over node i will not cause any energy costs due to the aggregation at node i.

That is, in order to construct an energy-efficient aggregation tree, each node i will propagate E_i^r, as in SEEF and MEEF, and modified energy cost $\bar{E}_i^e$, which is equal to

$$\bar{E}_i^e = \begin{cases} 0 & \text{if } i \text{ is a source node,} \\ E_i^e & \text{otherwise,} \end{cases} \tag{6.1}$$

to its adjacent neighbors. In so doing, the calculation of E_i^e given in Equation 4.21 of Chapter 4 will change to

$$E_i^e = \frac{\left(\varepsilon + \sum_{j \in \Omega_i} \rho_{i,\alpha(j)-1} \left(|\alpha(j) > 1|\varepsilon^{poll} + prr_{i,j}(\bar{E}_j^e + \varepsilon^{ack})\right)\right)\left(1 - \rho_{i,n}^{R+1}\right)}{1 - \rho_{i,n}}. \tag{6.2}$$

Due to these changes, we name these extended forwarding strategies *single-link* and *multi-link energy-efficient aggregation forwarding* (SEEAF and MEEAF), respectively.

6.3.2 The Problem of Forwarding Cycles

A problem that arises from the capability of aggregation is that now a node may have better energy efficiency than does one of its forwarding nodes. Thus, the energy efficiency will no longer monotonically decrease along a forwarding path. However, that was a mandatory requirement in EEF, to prevent the occurrence of forwarding cycles. Without such a monotonically decreasing function, establishing the aggregation tree will be really challenging if cycles must be avoided[2].

We will use the following heuristic: Rather than relying on the energy efficiency metric, each node will only consider those neighbors having a higher *end-to-end packet delivery ratio* than the node itself. As this metric will decrease with every intermediate forwarding node, it can be used to prevent forwarding cycles in the first place. However, in the case the delivery ratio is equal to that of a neighbor, the path length in terms of hops is used additionally to break ties.

[2]Due to the same reason, building an MST in a distributed way is not as simple as building an SPT.

Hence, E_i^{eff} will be calculated by taking only neighbors j into account that satisfy

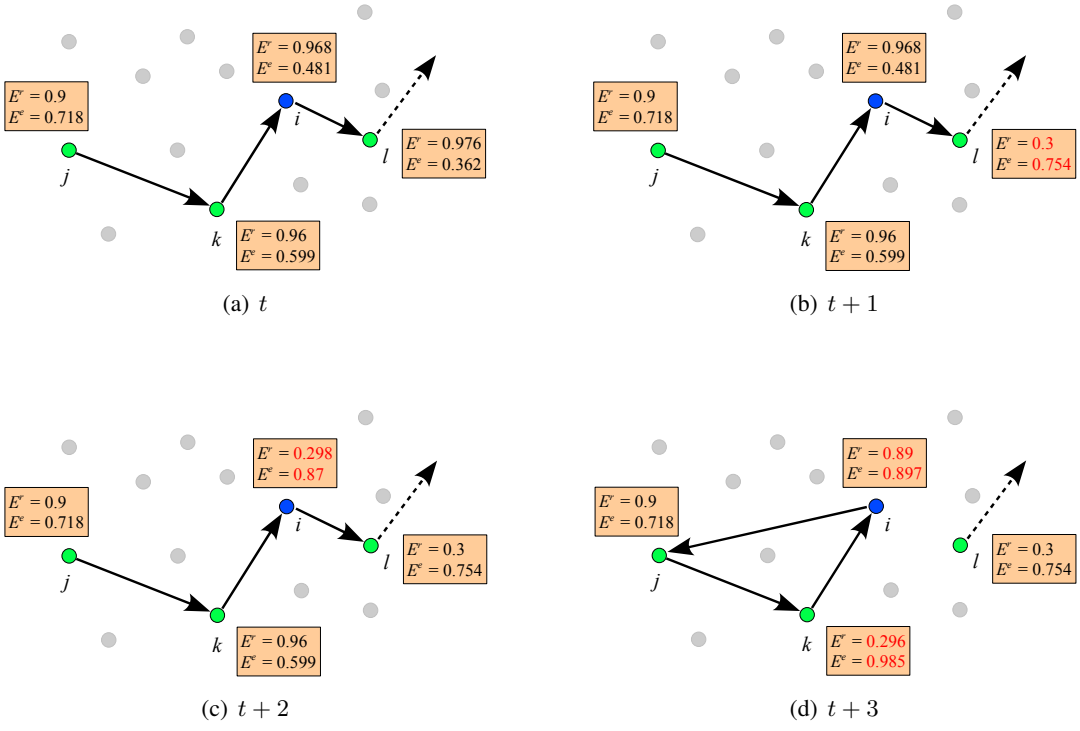
$$(E_i^r < E_j^r) \vee ((E_i^r = E_j^r) \wedge (E_i^h > E_j^h)). \tag{6.3}$$

E_i^h is the average path length of the forwarding, which is calculated similarly to Equation 4.21 of Chapter 4 as

$$E_i^h = \frac{\sum_{j \in \Omega_i} \rho_{i,\alpha(j)-1} prr_{i,j}(E_j^h + 1)}{\sum_{j \in \Omega_i} \rho_{i,\alpha(j)-1} prr_{i,j}}. \tag{6.4}$$

However, forwarding cycles may still occur if *outdated* information is used, i. e., a node uses E_j^r, and E_j^e, although they might have been changed. Since all nodes act simultaneously, and it can take some time until changes are propagated throughout the network, a node may be using information about forwarding paths that is no longer up-to-date.

For example, consider the case depicted in Figure 6.1, in which a node will run into a forwarding cycle, although each node has fulfilled requirement 6.3. The figure depicts a sample network at different times, or rounds, t. Each round, all nodes will propagate their forwarding variables to their neighbors, and we will assume that no packet losses occur.



(a) t (b) $t+1$

(c) $t+2$ (d) $t+3$

Figure 6.1: Illustration of a forwarding cycle

Figure 6.1(a) shows a part of the forwarding tree at time t, indicating the current end-to-end delivery ratios and energy costs for four nodes $i \ldots l$. In Figure 6.1(b), the end-to-end delivery ratio, as well as the energy cost, of the forwarding node l has worsened, which is propagated to node i at time $t+1$. The changes about the forwarding path have been processed by node i at time $t+2$, as shown in Figure 6.1(c), and will be further propagated to k. However, node j will not know about these changes

until $t + 4$. Thus, at $t + 3$, j will satisfy requirement 6.3 of node i due to a better end-to-end delivery ratio, and will be selected as the new forwarder of i. As shown in Figure 6.1(d), nodes i, j, and k will then form a forwarding cycle, which has been caused due to the usage of outdated information.

This problem can be solved by means of sequence numbers that indicate how up-to-date the information about forwarding paths is. The general idea is that the sink of the network periodically generates an increasing sequence number that is propagated throughout the network, together with the forwarding information contained in a beacon. Nodes receiving a beacon will update their own sequence numbers accordingly. However, with respect to the forwarding tree rooted at the sink, the sequence number will only be updated if a higher one is received from the current forwarding node.

If the forwarding path of a node has changed, the node will set a "timestamp", which is equal to its current sequence number. Thus, the next time the node recalculates its energy efficiency, it will only consider neighbors that satisfy requirement 6.3 and in addition have a higher sequence number than that set at the time the node's efficiency last changed. In this way, we can guarantee that each node which is used as a forwarder by a node i will not be a child of i in the forwarding tree.

However, we still must ensure that the sequence numbers will decrease along a forwarding path, i. e., that the sequence number used by a node will always be smaller than that of its forwarders. Hence, we must account for the case in which a node j will become the forwarder of a node i that has previously received a larger sequence number over another path. To overcome this problem, the "timestamp" is set by using the *highest* sequence number a node has ever received, rather than by using the sequence number currently in use.

6.3.3 An Algorithm to Prevent Forwarding Cycles

The algorithm to prevent forwarding cycles then works as follows: For each node i, we need three sequence numbers, which are denoted by s_i, s_change_i, and s_max_i. The sequence number s_i is propagated by i to all children in the forwarding tree by means of beacons. Each time a node receives such a sequence number from a parent node j, it will update its own sequence number by setting $s_i = s_j$. However, since in MEEAF each node might have several parents (forwarders), the node will set its sequence number to

$$s_i = \min_{\forall \text{parents } j} \{s_j\} \tag{6.5}$$

in this case. In addition, it will update its maximum sequence number s_max_i to

$$s\_max_i = \max_{\forall \text{parents } j} \{s\_max_i, s_j\}. \tag{6.6}$$

By using s_max_i, the third sequence number will be set to

$$s\_change_i = s\_max_i, \tag{6.7}$$

each time the forwarding path of node i or its energy efficiency has changed.

With these different sequence numbers on hand, we can formulate the second requirement that each forwarding node j must satisfy by requiring

$$s_j \geq s\_change_i. \tag{6.8}$$

In so doing, we can guarantee that each node i will only consider neighbors whose forwarding information is up-to-date, which will prevent nodes from running into cycles during the establishment of a forwarding path.

Figure 6.2 illustrates how the forwarding cycle from Figure 6.1 will then be prevented if sequence numbers are used. In each round, all nodes i will increase their current sequence number s_i by one and set s_change_i to s_max_i as soon as any changes have occurred. However, at $t+3$, node i will ignore the forwarding values of neighbor j due to requirement 6.8. Its forwarding link to node l will thus persist. Finally, at $t+4$, j will also notice the change in its energy efficiency and set its sequence number s_change to 7, as all other nodes have done before.

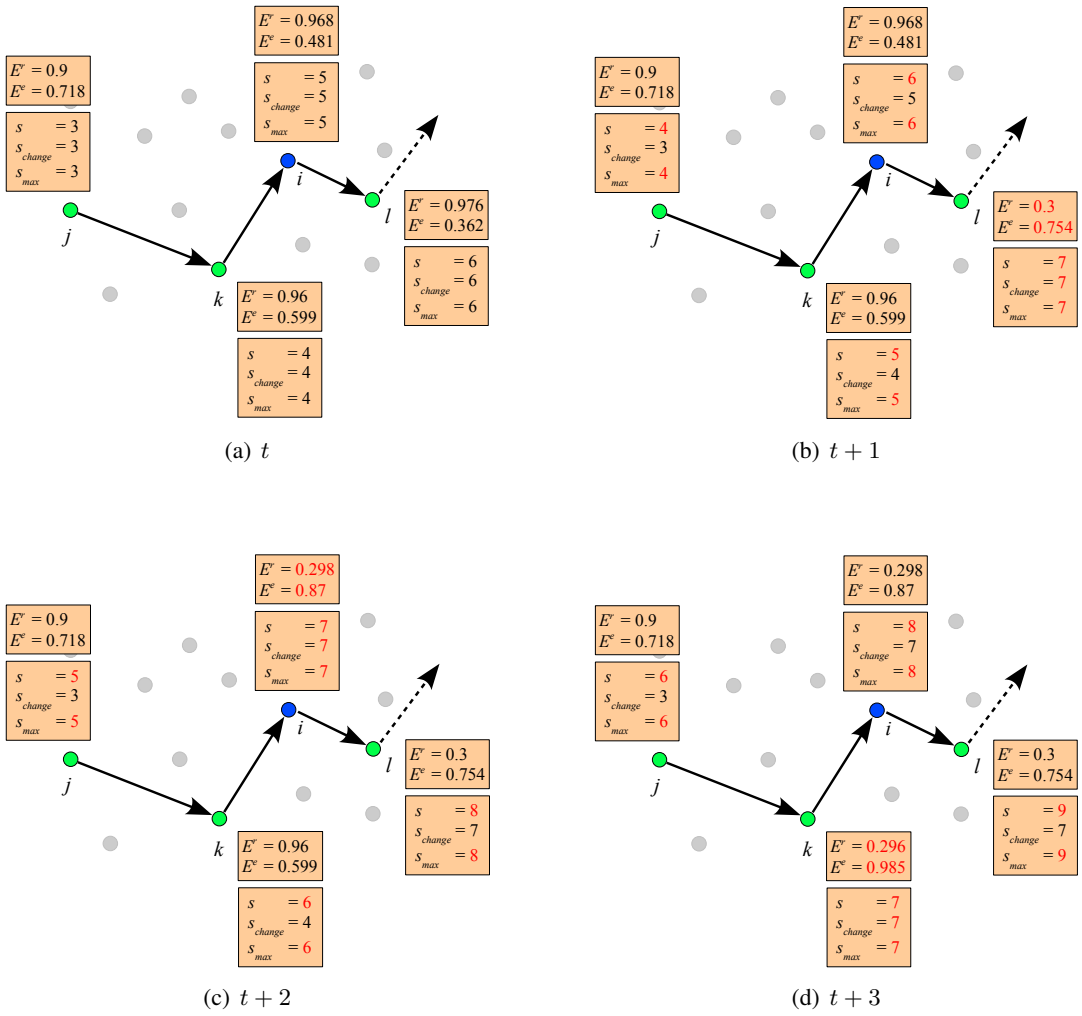


Figure 6.2: Prevention of forwarding cycles

6.3.4 The EEAF Algorithm

The complete EEAF algorithm is shown as Algorithm 6.1 for an arbitrary node i that has received a beacon from a neighbor j. The beacon will contain information regarding the current sequence number s_j of node j, its expected packet delivery ratio E_j^r, its energy cost E_j^e, and its expected path length E_j^h.

The energy efficiency of node i is then calculated for a single forwarder (SEEAF) or for a set of forwarding nodes (MEEAF) as follows: If j is a direct parent of node i, i first updates its sequence numbers s_i and s_max_i (lines 1-4). It then checks whether j is a new neighbor or whether the forwarding variables of j have been changed since the last beacon. If so, the information regarding the forwarding path of j will be updated (line 6). Additionally, i will recalculate its energy efficiency (lines 7-27) by considering all potential forwarding sets[3]. However, due to the conditions in lines 11 and 15, only non-child nodes are taken into account. After all forwarder sets have been considered, the one that maximizes the node's energy efficiency will be stored for later use, together with the forwarding parameters E_i^r, E_i^e, and E_i^h. Finally, i will update its sequence number s_change_i in order to exclude out-of-date neighbors if its parent(s), its end-to-end delivery ratio, or its energy cost of its forwarding path have changed (lines 28-30).

It should be noted that in dense networks it might be quite expensive to examine *all* forwarding sets. As in EEF and LEF, we have thus again limited the number of forwarding nodes, which can be selected at once, to three (line 9). While this has improved the computational complexity substantially, the performance of multi-link forwarding was actually not affected, as the numerical simulations in Section 4.5.5 of Chapter 4 had already indicated.

6.3.5 Further Discussions

Finally, we discuss how nodes may learn about the fact that they should act as source nodes. The easiest way to determine source nodes is to use the query disseminated by the network sink to get the desired information. Each node fulfilling the requirements to answer a query is then considered as an information source. For example, consider a temperature query used to obtain the average temperature in a network. Each node with a temperature sensor then serves as a source for answering it. If all nodes are equipped with such a sensor but only a fraction of α nodes should be used, sources can be distributed throughout the network randomly. Thus, a node will only issue data packets if $p \leq \alpha$, with p being a uniformly distributed random variable.

The issue becomes more complicated if the decision cannot be made beforehand, e. g., because a query depends on environmental events or stimuli. In such a case, a node can only use a *prediction value* denoted as β, which forecasts specific events based on information obtained from the past. As a heuristic, the expected energy cost E_i^e determined in line 26 of Algorithm 6.1 is then calculated according to

$$E_i^e = (1 - \beta)\hat{E}_i^{e*}. \tag{6.9}$$

[3]Note that for single-link forwarding, the forwarder set will only consist of one neighbor.

Algorithm 6.1 Receive EEAF message(j, s_j, E_j^r, E_j^e, E_j^h)

if $j \in parents_i$ **then**
  $s_i \leftarrow s_j$
  $s\_max_i \leftarrow \max\{s\_max_i, s_j\}$
end if
if change in j's forwarding variables **then**
  store j's new forwarding variables
  $\hat{E}_i^{r*} \leftarrow 0$
  $\hat{E}_i^{e*} \leftarrow \infty$
  $\hat{E}_i^{h*} \leftarrow 0$
  for all forwarding sets Ω_i: $|\Omega_i| = 3$ **do**
    if $\forall \hat{j} \in \Omega_i : (s_{\hat{j}} \geq s\_change_i) \wedge ((E_i^r < E_{\hat{j}}^r) \vee ((E_i^r = E_{\hat{j}}^r) \wedge (E_i^h > E_{\hat{j}}^h)))$ **then**
      calculate $\hat{E}_i^r$ according to Equation 4.19 from Chapter 4
      calculate $\hat{E}_i^e$ according to Equation 6.2
      calculate $\hat{E}_i^h$ according to Equation 6.4
      if $\forall \hat{j} \in \Omega_i : (\hat{E}_i^r < E_{\hat{j}}^r) \vee ((\hat{E}_i^r = E_{\hat{j}}^r) \wedge (\hat{E}_i^h > E_{\hat{j}}^h))$ **then**
        if $\hat{E}_i^r / \hat{E}_i^e > \hat{E}_i^{r*} / \hat{E}_i^{e*}$ **then**
          $\hat{E}_i^{r*} \leftarrow \hat{E}_i^r$
          $\hat{E}_i^{e*} \leftarrow \hat{E}_i^e$
          $\hat{E}_i^{h*} \leftarrow \hat{E}_i^h$
          set forwarders according to Ω_i, i. e., $parents_i \leftarrow \Omega_i$
        end if
      end if
    end if
  end for
  $E_i^r \leftarrow \hat{E}_i^{r*}$
  $E_i^e \leftarrow \begin{cases} 0 & \text{if } i \text{ is a source node} \\ \hat{E}_i^{e*} & \text{otherwise} \end{cases}$
  $E_i^h \leftarrow \hat{E}_i^{h*}$
  if forwarders or forwarding variables have changed **then**
    $s\_change_i \leftarrow s\_max_i$
  end if
end if

Another issue that should be considered regards the forwarding process itself. Saving energy in the network is only possible, if in-network processing really takes place. That means that a node should first gather all information from its children before forwarding the aggregate upwards the tree towards the sink. Otherwise, full aggregation is not possible. Thus, the forwarding process needs to be synchronized somehow. Therefore, each node could include information regarding its direct parents within the forwarding tree into its beacon messages. In so doing, all nodes would learn about their direct children. Nodes having no children could then issue and forward data packets immediately. Other nodes would have to wait a predefined time for packets to be received from their children. Once the timer has expired or all children have sent their information, an aggregation packet is created and in turn sent upwards the tree. Thus, the length of the waiting time defines the trade-off between the delay until queried information will reach the sink and the fraction of aggregation that will be achieved.

For the simulations and experimental experiments, which we will present in Sections 6.5 and 6.6, we have assumed that the sink has queried periodic information from a predefined fraction of source nodes. The waiting time was set to 30 seconds, which limited the maximum delay accordingly and offered full aggregation, because in the majority of cases forwarding packets arrived in time.

6.4 Other Aggregation Tree Constructions

In addition to SEEAF and MEEAF, we have also considered two other aggregation tree constructions. The first one is based on a greedy increment tree; the second one on a classic minimum spanning tree. Both approaches are only used for comparison because they do not scale well with the number of source nodes and thus are not recommendable for practical implementations. For completeness, we also considered an approximation of the Steiner minimum tree that was proposed by Kou *et al.* [134]. However, since the performance gains of their algorithm were marginal in our simulation setting, we have not included the results in Sections 6.5 and 6.6.

6.4.1 Greedy Increment Tree

An easy way to tackle the Steiner tree problem is to use a greedy approach. At first, no node in the network is aware of any kind of aggregation. The algorithm starts by carrying out a shortest path algorithm, with the link cost being equal to the energy efficiency according to Equation 4.22 from Chapter 4. The aggregation tree is then constructed by adding the best source node, in terms of efficiency, to the still empty tree (together with all nodes belonging to the appropriate path). In the next step, each node will again calculate its shortest path to the sink. However, this time, nodes already belonging to the aggregation tree will cause no energy costs. Thus, in this case, the energy efficiency is computed by using Equation 6.2 with

$$\bar{E}_i^e = \begin{cases} 0 & \text{if } i \text{ belongs to the aggregation tree,} \\ E_i^e & \text{otherwise.} \end{cases} \tag{6.10}$$

After each node has determined its new energy efficiency, the next source node not yet covered by the tree will be added. This procedure is repeated until all source nodes belong to the aggregation tree. Due to the greedy property of the algorithm, the tree is called *greedy increment tree* (GIT) [234].

In order to easily construct the GIT in a sensor network, the construction is handled by the network sink in a centralized fashion. At the beginning, the sink triggers each node, to determine its energy efficiency like in SEEF and MEEF. After that, it will gather the energy efficiency values from all source nodes in order to identify the most energy-efficient node. By sending an appropriate *add* packet to this node, all nodes along the path will be informed that from that point on they belong to the aggregation tree. Accordingly to Equation 6.10, these node will then update their energy cost metrics. In the next round, the sink again will trigger the remaining nodes to calculate their shortest paths until no uncovered source node remains.

Although this approach may be applicable for small sensor networks, it is not suitable for larger networks as it does not scale with the number of sources. Moreover, changes in the network are not easy to handle since the aggregation tree must actually be completely rebuilt. However, after it is constructed, we expect that the GIT will achieve a high energy efficiency: that is why it will be quite useful for comparison.

6.4.2 Minimum Spanning Tree

The minimum spanning tree may also be a good approximation of the optimal aggregation tree. It is defined as the tree that covers all nodes in the network with minimum link costs, which can be computed in a centralized manner by the well-known algorithms by Kruskal [136] or Prim [193]. However, constructing the MST in a distributed fashion is very expensive in terms of packet transmissions and time, although several approaches exist that have tried to solve this problem [16, 86, 91, 188],

As we are only interested in a performance comparison, we have used the following, centralized approach: First, we calculated the best paths between all pairs of nodes in terms of energy efficiency. The efficiency of these paths then served as edge costs for an overlay graph that consists of the sink as well as of all source nodes. Two nodes in this graph were thus connected by an edge if a path between them existed in the underlying network. For this overlay graph, the minimum spanning tree was constructed. In a final step, the edges in the spanning tree were then mapped to corresponding network paths.

It should be noted that the MST does not consider the energy efficiency on a path towards the network sink, but rather the energy efficiency between each pair of nodes. Nevertheless, its overall energy efficiency is expected to provide an almost optimal upper bound.

6.4.3 Steiner Minimum Tree Approximation

Another approximation of the SMT was proposed by Kou *et al.* [134], which is based on the complete overlay graph that consists of all source nodes and the sink. The algorithm works in five steps, which can be shortly described as follows:

1. Based on the shortest paths between all source nodes (including the sink), compute a complete overlay distance graph G_1.
2. Compute a minimum spanning tree on $G_1 \rightarrow G_2$.
3. Replace each edge in G_2 by one from G_1 with equal cost $\rightarrow G_3$.
4. Compute a minimum spanning tree on $G_3 \rightarrow G_4$.
5. Remove all leaves from G_4 that are neither the sink nor source nodes.

It has been shown that Kou's approximation has a cost of at most $2(1 - 1/n)C_{opt}$, with n being the number of source nodes and C_{opt} being the optimal cost of the SMT. However, in our simulation

setting, the performance achieved by GIT and MST in terms of the end-to-end energy efficiency has only been improved marginally. Due to its huge computational overhead, we thus excluded it from the simulations results, which we present in the following section.

6.5 Simulations

Based on the simulation setup used in Chapters 4 and 5, we have simulated the following forwarding strategies: MEEAF and SEEAF, MEEF and SEEF, MST, GIT, LPT [144], and GIST [125]. While LPT was implemented as described in [144] and thus relied on a hop-based forwarding strategy, GIST's quad-tree construction was slightly improved to take energy efficiency into account: Like in EEF and EEAF, packets were forwarded to a destination according to the best energy-efficient path found, which was used to establish the quad-tree. Thus, packets were not forwarded to the sink directly, but to intermediate nodes that acted as aggregation points. Upon aggregating all packets received from nearby nodes, these nodes then forwarded the aggregated information further to their own parents upwards the tree, until the actual network's sink was reached.

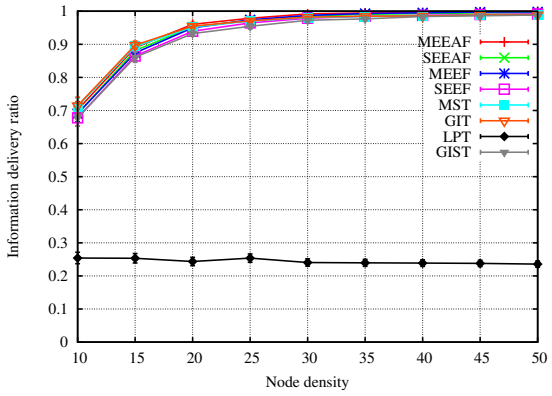
We simulated all forwarding strategies for different network densities and numbers of source nodes. For each setting, the simulations were repeated 500 times such that each data point in the following graphs indicates the average value over 500 runs. We assumed that all nodes were aware of their neighbors' packet reception ratios and knew whether or not they should act as a source node.

Like in Chapter 5, source nodes then periodically issued data packets, which were forwarded to one fixed sink node throughout the network. In order to investigate the influence of aggregation, we assumed that data packets were fully correlated and could be aggregated to one packet without the need of any extra space. As before, the propagation time on a link as well as the node's processing time were neglected. Also, network congestion was not considered. The number of retransmissions after data packets were discarded was set to three.

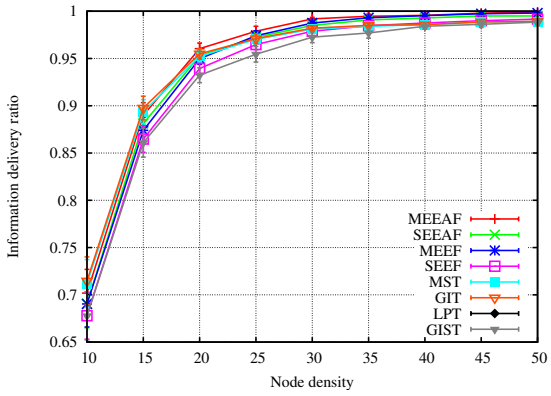
6.5.1 Influence of Node Density

The influence of the node density on different performance metrics is shown in Figure 6.3 for a source fraction α of 20%. Except for the LPT strategy that did not account for any packet reception ratios but only for the nodes' residual energy, all strategies achieved quite a high ratio of delivered information. Since data packets issued by source nodes will probably be aggregated along their forwarding paths, only one aggregation value might eventually reach the network sink. Thus, rather than depicting the delivery ratio of issued packets, Figures 6.3(a) and 6.3(b)[4] show the average so-called *information delivery ratio*, which is defined as the fraction of nodes used to calculate an aggregated value contained in a packet that finally arrives at the sink. Thus, the information delivery ratio gives an indication on the accuracy of the aggregation.

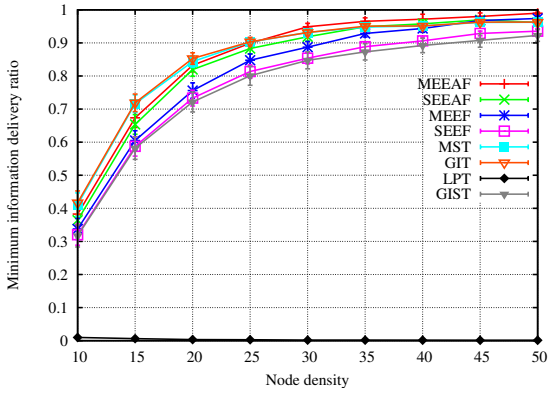
[4]Figure 6.3(b) shows a magnified version of Figure 6.3(a) in order to easier distinguish between the delivery ratios achieved by the different strategies.



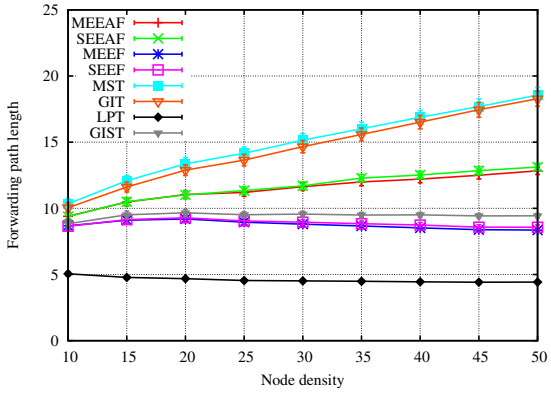
(a) Information delivery ratio



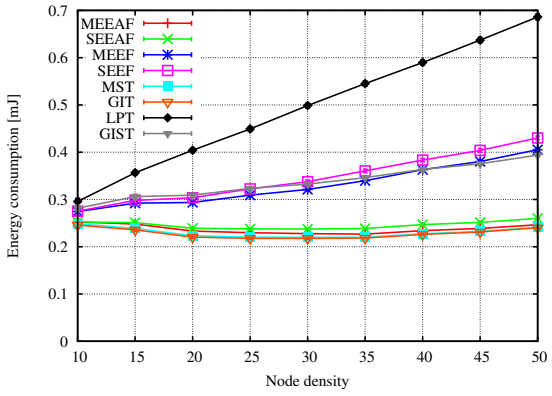
(b) Information delivery ratio (magnified)



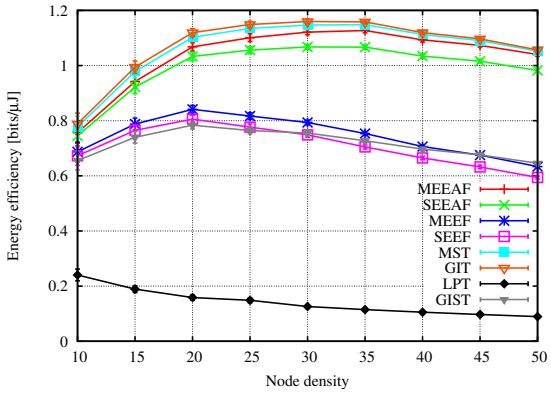
(c) Minimum information delivery ratio



(d) Forwarding path length



(e) Energy consumption



(f) Energy efficiency

Figure 6.3: Influence of node density ($\alpha = 0.2$, $R = 3$)

Except for LPT, which achieved an information delivery ratio of only 25%, the delivery ratios of all strategies differed by less than 5%. Among these strategies, GIST performed worst because data packets needed to be sent to intermediate nodes according to the established quad-tree, and could not be sent to the sink directly. MEEF and SEEF thus performed somewhat better. Unlike GIST, neither took energy savings due to aggregation into account[5]. Thus, intermediate aggregation nodes like in

[5] At least not during the construction of the forwarding tree

GIST were not required, which led to forwarding paths that were slightly shorter (see Figure 6.3(d)). As a consequence, the end-to-end delivery ratio of EEF was higher than that of GIST.

However, taking advantage of aggregation *a priori* improved the ratio of delivered information even more, as shown in Figure 6.3(b) for EEAF, MST, and GIT. Although those strategies established the longest forwarding paths (Figure 6.3(d)), they performed best in terms of delivered information. Though this seems to be odd at a first glance, it turned out that the reception ratios on their forwarding links were significantly better. Hence, EEAF, MST, and GIT were able to establish mainly short-distance links, which increased the forwarding path length but at the same time improved the end-to-end information delivery ratio.

In contrast, EEF explicitly avoided establishing too many short-distance links, for efficiency reasons. As aggregation is not taken into account by EEF a priori, it was not aware of any energy savings that could be exploited if packets were sent along aggregation paths. Thus, its calculated energy consumption would have been substantially higher if forwarding paths had been composed of mainly short-distance links. On the other hand, EEAF took advantage of that fact. Because source nodes informed adjacent nodes that they had zero energy costs, they even "attracted" forwarding paths from their neighborhood. In so doing, longer paths did not worsen the energy efficiency of nearby nodes but even improved it due to better delivery ratios. This behavior is also shown in Figure 6.3(c), which depicts the information delivery ratio of the worst-connected source node in the network.

As we have already seen in Chapters 4 and 5, multi-link forwarding is able to further improve the delivery performance. As shown in Figures 6.3(b) and 6.3(c), this is also true in the case of aggregation. Both MEEAF and MEEF strategies clearly outperformed SEEAF and SEEF, respectively. At the same time, multi-link forwarding slightly shortened the average forwarding path length, as shown in Figure 6.3(d), and caused fewer retransmissions than did single-link forwarding.

Figure 6.3(e) shows the network's energy consumption for all strategies, averaged over the number of source nodes. As expected, MST and GIT performed best and consumed the least energy, which is mainly due to the fact that they relied on the minimum spanning tree, respectively on the greedy increment tree. The energy consumption of EEAF was only slightly higher, considering the fact that it operated in a fully distributed way without using global knowledge. On the other hand, LPT performed worst. Much energy was consumed to retransmit data packets, although significantly fewer nodes were involved in the forwarding of packets. The performance of EEF and GIST was (unexpectedly) quite similar. Thus, in this simulation setting, GIST could not benefit from its quad-tree as expected. Only at high node densities did it consume less energy than did MEEF. However, this might be different in larger networks, in which the hierarchical approach of GIST could be better exploited. But nevertheless, as long as the source nodes are known a priori, the *group-independent* aggregation tree of GIST is still expected to be outperformed by MEEAF, as well as by SEEAF.

The energy efficiency of each forwarding strategy is depicted in Figure 6.3(f), which illustrates the trade-off between the information delivery ratio and the energy consumption. As expected, the best efficiency was achieved by GIT, followed by MST. However, it should be noted that neither strategy can be recommended for use in large sensor networks, which may consist of hundreds or even thousands of nodes. The overhead spent to establish both aggregation trees will simply be too high.

Nevertheless, they are very suitable for comparison, as they provide an upper bound for other algorithms.

As Figure 6.3(f) shows, among all distributed algorithms considered, MEEAF, as well as SEEAF, performed best and almost reached the energy efficiency level found in GIT forwarding. Even though SEEAF was clearly outperformed by MEEAF due to a better delivery ratio and, additionally, lower energy cost, it showed considerable improvements over EEF and GIST. Because GIST consumed much energy (caused by its group-independence), it was not able to achieve a significantly better energy efficiency than did MEEF. However, the worst performance was achieved by LPT, as it suffered heavily from poor forwarding links that caused both a low delivery ratio and a high energy consumption.

6.5.2 Influence of the Number of Source Nodes

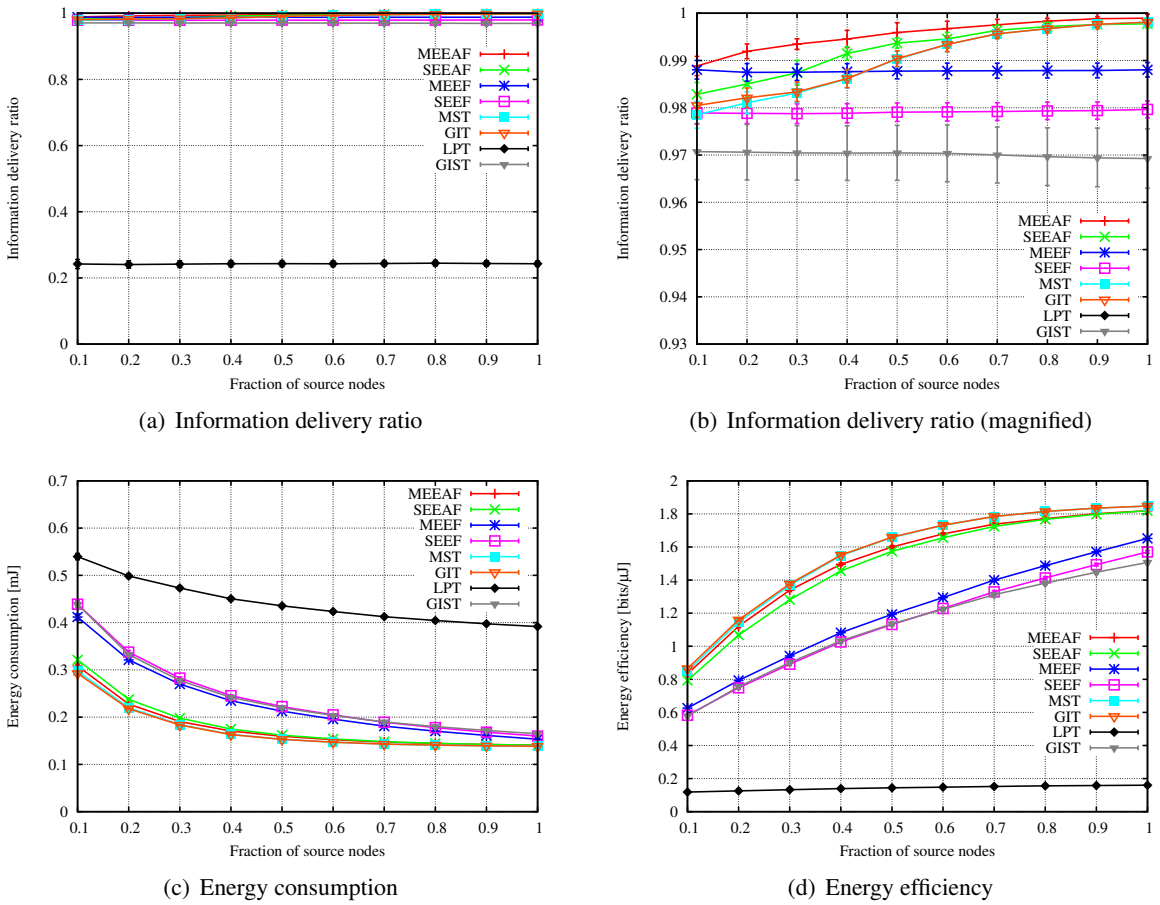
In addition to different node densities, we have also varied the number of nodes that issued data packets. We considered a network with a density of 30 nodes per maximum transmission range, and increased the source node fraction α from 0.1 and 1. The simulation results are depicted in Figure 6.4.

The impact on the average information delivery ratio is shown in Figure 6.4(a), respectively in Figure 6.4(b). The ratio of information delivered by EEF, LPT, and GIST did not change very much, even though the source fraction varied. This is due to the fact that the tree construction of these strategies did not depend on the number of source nodes and thus did not change. In contrast, EEAF, MST, and GIT were able to improve their performance for higher source fractions. More source nodes led to more aggregation points within the network, which was beneficial for each of them. Thus, if the number of source nodes increased, sending packets towards a nearby aggregation path that had a better end-to-end delivery ratio became more efficient, as due to aggregation it consumed less energy. Since EEF, as well as LPT and GIST, neglected such aggregation gains during the construction of their forwarding paths, they were not able to take advantage of them. Consequently, they could not benefit from longer, but better connected forwarding paths.

Although the total energy consumption within the network grew for a higher source fraction (due to more packets issued), Figure 6.4(c) illustrates that the *average* energy consumption on a forwarding path decreased. That is, the larger the number of sources, the larger the aggregation gain per source node was. The reasons are as before; more sources will increase the fraction of packets that can be aggregated, thus decreasing the average energy consumption.

A comparison of the energy consumption by all strategies shows a result similar to that in Figure 6.3(e): GIT, MST, and EEAF performed best, followed by EEF, and GIST. LPT again consumed the greatest amount of energy. However, for $\alpha \rightarrow 1$, we see that EEF almost achieved the same energy consumption as did EEAF. That is due to the fact that if every node acts as a source, data aggregation will also take place at every node. Thus, the benefit of explicitly forwarding data to a distant aggregation node almost vanished. As Figure 6.3(e) shows, forwarding data to the sink directly, as done by EEF, then did not perform much worse.

However, EEAF still performed considerably better in terms of energy efficiency, as is shown in Figure 6.4(d). Independent of the number of data sources within the network, EEF, GIST, and LPT were



(a) Information delivery ratio

(b) Information delivery ratio (magnified)

(c) Energy consumption

(d) Energy efficiency

Figure 6.4: Influence of the number of source nodes ($\mu = 30$, $R = 3$)

outperformed. For $\alpha \to 1$, EEAF almost reached even the upper bound of GIT and MST, which again achieved the best efficiency.

In conclusion, the simulation results indicate that the forwarding metrics used by EEAF are quite reasonable, showing the desired network performance if the source nodes are known *a priori*. However, in the case where source nodes cannot determined beforehand, using GIST or EEF may be a good alternative, as both performed considerably better than did LPT.

6.6 Experimental Evaluation

Like in Chapters 4 and 5, we also carried out real-world experiments and evaluated the performance of aggregation in our WSN testbed.

6.6.1 Experimental Setup

The experimental setup was as before. All nodes used a transmission power of 15% and established forwarding paths according to the strategy being evaluated. Then, five nodes were randomly picked as

source nodes, which issued 32-byte data packets at a rate of one packet per round (30 seconds), starting randomly in time. The maximum number of retransmissions was again set to three. The evaluation lasted for 100 minutes such that each source sent a total of 200 packets towards the sink.

Upon choosing the source nodes, the network's sink started to send periodic beacons and thereby triggered the construction of the aggregation tree (see Section 4.7 of Chapter 4). Again, link measurements had been carried out beforehand in order to get an estimate of the links' packet reception ratios. Once all forwarding paths had been established, the actual evaluation started. Data packets were issued by source nodes and sent to the first forwarding node. However, unlike in Chapters 4 and 5, they were not forwarded immediately but buffered for one round (30 seconds). In so doing, data packets could first be aggregated and then forwarded to the next hop. Thus, it took some time until the first aggregation packet had reached the sink.

6.6.2 Evaluation Results

Except for the MST strategy, which was not evaluated due to its high overhead to operate in a distributed manner, all other strategies were implemented, tested, and evaluated. The results of the experiments, which were repeated ten times, are shown in Table 6.1, by means of average values as well as appropriate 0.95 t-quantiles.

The first row of Table 6.1 shows the energy efficiency of each strategy, which basically confirms our simulation results from the previous section. Again, the greedy increment aggregation tree (GIT) achieved the highest energy efficiency and thus provided an upper bound for all other strategies. The energy efficiency of MEEAF was only 6% worse than that of GIT, and thus better than those of the remaining strategies. Also, SEEAF performed better than did MEEF, SEEF, LPT, and GIST. However, the improvement in EEF's performance in terms of energy efficiency was only about 6% to 7%. This is quite different from the simulation results, which showed an improvement of up to 60%. However, due to the smaller network size, the testbed offered significantly fewer forwarding alternatives than did the simulation setup. Thus, the benefit of EEAF decreased substantially, as several forwarding paths were equal to those of EEF.

Similar to the results we obtained by means of simulations, the LPT strategy performed worst and achieved a poor efficiency. It consumed more than twice the energy spent by EEAF and EEF because significantly more packet transmissions (tx data) occurred. As in the simulation, LPT suffered from long-distance but poor forwarding links, which is shown in the last two lines of Table 6.1 by the hop counter and by the number of retransmissions required. While the forwarding path length of LPT was around 40% shorter than those of other strategies, the number of retransmissions per link was more than five times higher. As a consequence, LPT achieved a delivery ratio of only 55%.

The total number of transmitted data packets, averaged over all nodes within the network, is shown in the second row of Table 6.1. Again, multi-link forwarding caused fewer transmission than did single-link forwarding, although more control packets needed to be sent to poll backup nodes (tx control). Thus, backup nodes were indeed useful and were exploited by MEEAF, as well as by MEEF, which is shown by the average number of received data and control packets (rx data and control). Although

Strategy	MEEAF	SEEAF	MEEF
Energy efficiency [bits/μJ]	0.8420 [0.6928, 0.9912]	0.8331 [0.6791, 0.9871]	0.7961 [0.6044, 0.9876]
Tx data	98.33 [80.06, 116.60]	101.44 [82.63, 120.26]	112.86 [83.25, 142.47]
Tx control	84.19 [61.29, 107.08]	75.03 [56.25, 93.80]	100.90 [69.20, 132.60]
Rx data	86.29 [60.01, 112.57]	83.86 [60.16, 107.56]	105.06 [67.95, 142.16]
Rx control	110.78 [77.85, 143.71]	87.56 [70.94, 104.17]	130.88 [85.61, 176.16]
Information units delivered	993.10 [980.36, 1005.84]	990.40 [975.86, 1004.94]	989.40 [972.91, 1005.89]
Information units delivered per source	198.62 [196.07, 201.17]	198.08 [195.17, 200.99]	197.88 [194.58, 201.18]
Min. information units delivered	193.30 [180.53, 206.07]	190.60 [176.01, 205.19]	189.40 [172.91, 205.89]
Energy consumption [mJ]	320.08 [260.56, 379.61]	323.07 [264.09, 382.05]	356.64 [263.11, 450.17]
Energy consumption per source [mJ]	64.02 [52.11, 75.92]	64.61 [52.82, 76.41]	71.32 [52.62, 90.04]
Max. energy consumption [mJ]	126.34 [93.39, 159.29]	120.05 [90.16, 149.93]	136.00 [95.88, 176.13]
Hop counter	2.87 [2.47, 3.28]	2.96 [2.48, 3.43]	2.66 [2.37, 2.94]
Retransmissions	0.21 [0.09, 0.33]	0.22 [0.11, 0.33]	0.29 [0.15, 0.43]

Strategy	SEEF	GIT	LPT
Energy efficiency [bits/μJ]	0.7804 [0.5956, 0.9651]	0.8940 [0.7159, 1.0721]	0.1826 [0.0996, 0.2656]
Tx data	113.93 [82.42, 145.43]	95.92 [74.81, 117.04]	297.14 [238.54, 355.74]
Tx control	90.19 [64.78, 115.60]	74.40 [57.48, 91.31]	65.14 [41.51, 88.76]
Rx data	100.12 [65.38, 134.85]	84.40 [61.12, 107.67]	164.14 [107.85, 220.42]
Rx control	101.08 [77.63, 124.53]	84.47 [69.00, 99.94]	85.46 [61.37, 109.55]
Information units delivered	989.60 [973.83, 1005.37]	989.80 [974.40, 1005.20]	553.30 [422.55, 684.05]
Information units delivered per source	197.92 [194.77, 201.07]	197.96 [194.88, 201.04]	110.66 [84.51, 136.81]
Min. information units delivered	189.60 [173.83, 205.37]	189.80 [174.40, 205.20]	10.30 [4.54, 16.06]
Energy consumption [mJ]	363.63 [264.94, 462.32]	305.99 [240.57, 371.41]	896.43 [722.52, 1070.35]
Energy consumption per source [mJ]	72.73 [52.99, 92.46]	61.20 [48.11, 74.28]	179.29 [144.50, 214.07]
Max. energy consumption [mJ]	140.30 [98.75, 181.84]	123.87 [80.97, 166.78]	336.55 [279.69, 393.42]
Hop counter	2.70 [2.41, 2.99]	3.30 [2.75, 3.85]	1.71 [1.52, 1.90]
Retransmissions	0.31 [0.10, 0.51]	0.21 [0.06, 0.35]	1.63 [1.37, 1.88]

Strategy	GIST
Energy efficiency [bits/μJ]	0.7806 [0.6308, 0.9304]
Tx data	109.56 [87.41, 131.72]
Tx control	76.68 [54.51, 98.85]
Rx data	85.64 [59.62, 111.65]
Rx control	92.56 [73.25, 111.87]
Information units delivered	991.40 [978.14, 1004.66]
Information units delivered per source	198.28 [195.63, 200.93]
Min. information units delivered	191.40 [178.14, 204.66]
Energy consumption [mJ]	348.35 [279.03, 417.67]
Energy consumption per source [mJ]	69.67 [55.81, 83.53]
Max. energy consumption [mJ]	138.41 [92.88, 183.93]
Hop counter	2.63 [2.30, 2.97]
Retransmissions	0.26 [0.16, 0.36]

Table 6.1: Results of the experimental evaluation (EEAF)

fewer data packets were sent, more packets were received by forwarding nodes. Furthermore, as indicated by the hop counter, backup nodes were often closer to the sink. Polling those nodes thus increased the likelihood of forwarding data successfully. At the same, the number of retransmissions was reduced.

The remaining rows of Table 6.1 show the amount of delivered information units and the appropriate energy consumption in the network, as well as the energy consumption caused per source node. Except for LPT, all strategies performed very well and achieved high delivery ratios. Thereby, the total energy consumption per source was between 60 and 70 mJ on average; for LPT about 180 mJ. As a result, all strategies (except for LPT) achieved an energy efficiency of more than 0.78 bits/μJ.

Comparing the results with those presented in Table 4.1 of Chapter 4 illustrates that data aggregation leads to significant improvements, in terms of the amount of delivered information, consumed energy, and energy efficiency. For example, the energy efficiency improved more than twofold. However, we must take into account that the experiments performed in Chapter 4 used a source node fraction of 100%, while Table 4.1 shows the result for only 20% of data sources. Thus, in this experiment, far fewer packets got lost due to network congestion. Nevertheless, we can conclude that aggregation will reduce the energy spent on transmissions significantly, which will affect both the packet delivery ratio and the network congestion beneficially.

6.7 Conclusions

In this chapter, we have analyzed the impact of data aggregation during the forwarding process in a wireless sensor network. Based on energy-efficient forwarding presented in Chapter 4, we proposed an extension that takes energy savings due to aggregation into account. In this way, other forwarding path get selected, improving the information delivery process, the energy consumption, and the energy efficiency.

The extension of EEF is easy to implement, as only the manner in which energy costs are propagated needs to be changed. However, care must be taken in order to avoid forwarding cycles. We thus proposed a simple set of sequence numbers which is used to identify outdated information from adjacent neighbors. In particular, our approach is designed to cover cycles caused by multi-link forwarding, where several forwarding paths need to be taken into account at the same time. The same set of sequence numbers can also be applied for EEF and LEF solely. Although both strategies try to prevent forwarding cycles by requiring the node's energy efficiency to be smaller than that of a forwarding node, cycles may occur as soon as packet reception ratios between nodes start to change. In this case, sequence numbers can be used similarly to EEAF.

The simulations, as well as the experimental evaluations, have shown that EEAF clearly outperforms EEF in terms of the information delivery ratio, consumed energy, and energy efficiency. Compared to the best results achieved by strategies based on a greedy increment tree (GIT) or minimum spanning tree (MST), EEAF performed very well and almost achieved the same results; the reader should keep in mind that it is a fully distributed algorithm. As for EEF and LEF, multi-link forwarding again

improved single-link forwarding in terms of delivered information as well as energy efficiency. Thus, MEEAF seams to be a very promising strategy for aggregation scenarios as considered in this chapter.

Nevertheless, a drawback of EEAF is perhaps the condition that source nodes must be known *before* the aggregation tree construction can start. Although this might often be possible, there might also be scenarios in which this information is not available. In this case, the group-independent spanning trees proposed by GIST seems to be a good alternative[6]. Although its energy efficiency was not significantly better than that of EEF, we believe that this will change if the network size increases. However, we leave this investigation to future work.

In conclusion, we now have a comprehensive framework of forwarding strategies on hand that accounts for energy efficiency, lifetime efficiency, and energy-efficient aggregation. As already mentioned earlier, which strategy is most suitable depends on the application scenario. In future work, LEF could be extended similarly to EEF by an aggregation component that would modify the energy cost of source nodes appropriately. In so doing, lifetime-efficient aggregation forwarding could be provided, too.

That concludes our analyses of the forwarding strategies contained in this thesis. In the following chapter, we will finally consider the potential of topology management in order to conserve energy in a different way. By putting redundant nodes into a low-power sleep mode, energy will be consumed only by active nodes, which form a connected overlay network that can be used for communication purposes.

[6] If it is extended like in this chapter to account for energy efficiency.

CHAPTER 7

A Topology and Energy Control Algorithm

"The greatest obstacle to discovery is not ignorance, but the illusion of knowledge."

– D. Boorstin –

7.1 Introduction

As research in sensor networks has become more and more widespread [11, 67], low-cost hardware and embedded systems have been enabled by micro-sensor and radio technology advances. As we have seen, by taking energy efficiency into consideration [127], densely populated sensor networks consisting of hundreds or thousands of battery-powered nodes are no longer unrealistic. However, recharging might be impossible due to local conditions or perhaps even inefficient due to low-cost components. Thus, in order to minimize the energy consumption and extend the network's lifetime, algorithms are also required that take advantage of high node densities to exploit available redundancy while still maintaining the network operable.

In this thesis, we have so far considered energy-efficient algorithms that are tailored to the problem of data forwarding in order to save energy. Unnecessary transmissions are avoided, either due to forward error correction, energy-efficient forwarding paths, or due to in-network processing. However, as most energy is spent on keeping the radio transceiver active [115, 211], even in idle mode, this issue requires further consideration. Instead of keeping the transceiver on all the time, all sensor nodes could save the most energy if they turned their wireless communication radios off most of the time. To still maintain communication through the network, it is necessary that either some or all nodes wake up periodically or at certain times. Alternatively, it is also possible that a small number of nodes is prevented from turning their radios off at all. It is quite challenging to make sure that the network is not partitioned. The decision as to which nodes may sleep and which nodes must be awake is the task of the *topology control* or *topology management* that is tackled in this chapter.

In the following, we will present a new *topology and energy control algorithm* called TECA that (i) extends the network's lifetime by putting nodes into sleep mode (radio is turned off), and (ii) still guarantees network connectivity. We will show by means of simulations that our algorithm saves more energy than other approaches and additionally minimizes packet losses by considering link qualities appropriately. Furthermore, network partitions are effectively avoided.

The structure of this chapter is as follows: In the next section, we first outline related work. After that we describe TECA in its basic functions and discuss some details in Section 7.3. An exploration of different parameter settings of TECA is provided in Section 7.4. A comparison with two other topology management approaches is presented in Section 7.5 by means of simulations. Section 7.6 contains an evaluation of real-world experiments. Finally, the chapter ends with concluding remarks in Section 7.7.

7.2 Related Work

Several energy-aware approaches have been proposed in the literature. In addition to the work described in Chapters 4 to 6, adaptive MAC layer protocols like S-MAC [270], T-MAC [241] and WiseMAC [81] allow nodes to turn their radios off if they are not participating in communication. All of these protocols direct their major focus to reducing *idle listening* by introducing a duty cycle. In S-MAC, time is slotted in relatively large time frames consisting of an active and a sleep cycle. Only in the active cycle are data transmissions possible. During the sleep cycle, the radio is turned off to save energy. The length of the active cycle is fixed, whereas the sleep time is under the control of the application. As a consequence, nodes must listen to the channel during the entire active phase, even if no transmissions are detected. T-MAC improves this situation by introducing a timeout value TA that determines the length of the active cycle. If a node detects activity on the channel after the timeout (data transmissions or collisions), it resets the timer. Otherwise, it will assume that the channel is idle and will go to sleep. Unlike S-MAC and T-MAC, WiseMAC is based on preamble techniques, while active and sleeping phases are not synchronized among nodes. Rather, each node informs its neighbors about its own wake-up time. With this knowledge, another node that has data to send just starts the transmission at the right time with a wake-up preamble.

In addition to energy-efficient MAC layer protocols, there exist several algorithms in the literature regarding topology control or topology management. Typically, two different approaches can be distinguished: (i) power control algorithms that minimize the node's transmission power, and (ii) topology-based approaches that build a backbone of active nodes, which perform data forwarding while all other nodes turn their radios off. Both approaches are orthogonal to the above-mentioned energy-efficient MAC protocols and can be combined easily.

The goals of power control approaches are to minimize interference, packet collisions and retransmissions, as well as to improve spatial reuse[1] [235, 268]. Ramanathan *et al.* [197] present two algorithms that maintain connectivity in the network by adjusting the maximum transmission power of a node. Both algorithms are centralized and based on global knowledge. Li *et al.* [151, 152] propose LMST,

[1] Spatial reuse refers to the scheduling of multiple (mutually non-interfering) transmissions simultaneously.

where each node builds a minimum spanning tree based on local information. The authors prove that their topology control algorithm preserves network connectivity, while the node degree is bounded by six. Furthermore, the constructed topology can be transformed into a bidirectional one by removing all uni-directional links without affecting connectivity. While the algorithm is limited to homogeneous networks, Li and Hou extend their approach in [150] to heterogeneous networks where nodes may have different maximum transmission ranges.

Another approach that relies on the number of adjacent neighbors is proposed by Blough *et al.* [27]. In k-Neigh, each node adaptively decreases its transmission power until just k symmetric links to adjacent nodes remain. Using distance estimates, the k nearest neighbors are determined that then control the node's transmission power. The authors prove that their algorithm terminates after a total of $2n$ messages have been exchanged, with n being the number of nodes in the network. Also, they give an estimate for k that achieves connectivity with high probability. Interestingly, the value of k is only loosely dependent on the number n of nodes. Thus, knowledge about an exact value for n is not necessary. However, implementing k-Neigh in practice requires the ability to make good distance estimates, which is not always possible.

Although many approaches claim to reduce interference by means of the spareness of the network as a result of power adjustment, Burkhart *et al.* [33] disprove this implication. They propose interference-minimal connectivity-preserving and spanner constructions. However, power control approaches may improve the network's capacity by reducing interference, but in terms of energy efficiency, their savings will likely be marginal compared to those achieved by reducing idle listening [213]. In contrast to power control, topology-based approaches exploit the fact that in densely populated networks many nodes will be redundant. By turning the radios of these nodes off, a topology of active nodes is constructed. Because idle listening is omitted, those approaches will likely yield much better energy conservation.

Xu *et al.* [262] have proposed *geographic adaptive fidelity* (GAF), a topology control protocol that uses the geographic positions of nodes to subdivide the sensor network into virtual grid cells. The cell size of the grid is chosen such that all nodes in one cell are able to communicate with all other nodes in adjacent cells. Given a transmission range r, the cell size a is equal to $r/\sqrt{5}$. Since from a routing perspective nodes in the same grid cell are considered equivalent, just one node needs to be active with its radio turned on, while all other nodes may sleep. In so doing, GAF exploits the redundancy in the network and prolongs the network's lifetime with increasing node density. To balance the energy consumption, sleeping nodes periodically wake up and rotate the role of the awake node among themselves. Since GAF assumes that each active node can communicate with all nodes in adjacent cells, the network is assumed to remain connected by the built backbone topology, i. e., there will be no more partitions than in the underlying raw topology. However, this assumption does not always hold true in reality, where nodes might experience high packet losses. Furthermore, the geographic information GAF relies on is not always available.

Similar to GAF, Ye *et al.* [269] have proposed PEAS, a robust energy-conserving protocol for long-lived sensor networks. However, no geographic information is required. Instead, nodes use two transmission powers for the communication within a transmission range r_t and a probing range r_p. Like grid cells in GAF, only one node per probing range has to be turned on all the time, while all

other nodes sleep. For $r_t \geq (1+\sqrt{5})r_p$, PEAS then achieves asymptotic connectivity, using the same assumptions as GAF.

Span [54, 55] is another protocol that builds a topology backbone composed of forwarding "coordinators". It attempts to preserve the network's original capacity and connectivity while reducing its energy consumption. The node's decision to become a coordinator depends on its residual energy and its surrounding neighborhood. For example, a node will become a coordinator if it is able to connect two other coordinators that were not yet connected. Like in GAF and PEAS, nodes will periodically wake up to balance the energy consumption, but will sleep if they are redundant regarding the forwarding backbone.

TMPO [21] has many concepts in common with Span. However, its focus is on efficient communication rather than on energy conservation due to sleeping nodes. Based on a *minimal dominating set* (MDS), a backbone topology is constructed by transforming the MDS into a *connected dominating set* (CDS) [93]. TMPO uses the concept of clustering [10, 104] to build the MDS. By introducing *gateways* and *doorways*, these clusters get connected in the CDS, guaranteeing network connectivity. A similar approach termed *cluster-based energy conservation* (CEC) is presented in [260].

Nikaein and Bonnet [183] emphasize a similar motivation and focus on the routing performance. They propose an algorithm that constructs a routing topology based on a forest. Each tree in the forest forms a zone that is maintained proactively. In so doing, the network can be seen as a set of non-overlapping zones, which are linked by extracting the best nodes in terms of connectivity. The authors have shown that the routing performance is significantly improved with the help of topology management, in terms of packet delivery ratios as well as delays.

Dousse *et al.* [77] consider networks where nodes switch between *on* and *off* modes independent of each other. Since the on/off schedules are completely uncoordinated, the network might be disconnected most of the time. Assuming a store-and-forward routing mechanism, data is sent from nodes to a sink. Under some simplifying conditions, the maximum latency and variance are bounded. Moreover, the latency grows linearly with the distance between a node and a sink, depending on the node density, the connectivity range, and the duration of active and sleep periods.

Also putting nodes to sleep, ASCENT [50] builds a topology relying on the number of neighbors a node has discovered. However, only those nodes whose packet losses are below a given threshold are considered to be neighbors. The neighbor threshold NT determines whether or not a node will join the backbone topology. If the node's number of neighbors is smaller than NT, the node will become active with its radio turned on. Otherwise, ASCENT will assume that there are enough active neighbors maintaining connectivity, and the node will switch to a passive state. Although passive nodes do not join the backbone topology, they still overhear packet transmissions. If the number of neighbors drops below NT, or active nodes send help messages indicating poor link qualities to adjacent nodes, a passive node will revert to the active state. Since ASCENT attempts to prolong the network's lifetime, each node will have a passive timeout after it goes to sleep and turn off its radio. In addition, there will be a sleeping timeout after sleeping nodes have moved back to the passive state again. However, due to the simplicity of ASCENT and the fact that it mainly relies on the number of active neighbors, there might be situations where the network is partitioned.

Like ASCENT, Naps [98] bounds the node degree in the constructed topology. While ASCENT tries to achieve a stable system, Naps puts nodes to sleep faster and more aggressively. Simulations show that most of the nodes are part of the largest connected component, but several distinct partitions are not unlikely. Similar to Naps, AFECA [261] puts nodes into sleep mode, with the sleeping time being related to the number of neighbors. Hence, each node must have an accurate knowledge of its neighborhood's size.

Gupta *et al.* [103] studied the problem of the minimum *connected sensor cover*: In response to a query, the network is self-organized into a logical topology that involves only a small subset of sensor nodes that are sufficient to process the query. Other nodes do not participate during the query execution and thus are able to save energy. However, they must not necessarily turn their communication radios off. The minimum connected sensor cover then describes the minimum subset of nodes that are connected and that cover the region specified in the query. Of course, it will be more beneficial to construct the topology if the query runs sufficiently long. Otherwise, the communication cost of constructing the topology may not be amortized by its energy savings.

Schurgers *et al.* [214] propose STEM, a topology control protocol where nodes are equipped with a second radio channel for paging. The paging channel operates on a lower frequency and with less bandwidth in order to conserve energy. In idle operation, nodes will shut down their main radios, while their second radios will remain active all the time. In case packets need to be forwarded, the paging channel will be used to turn the main radio on to enable data communication.

In contrast to related work, we present a topology-based approach without the need of a second communication radio or the knowledge of geographic information. Unlike most of the proposed algorithms, our simulations are based on a realistic connectivity model, which considers different packet reception ratios rather than using a simplified unit disk graph. We assume that all nodes will be battery-powered and will have the same transmission range. Mobility is not taken into account since we assume that most applications will rely on static sensor network.

7.3 The Topology and Energy Control Algorithm

Before we compare our approach with GAF, ASCENT, and a randomized algorithm called RAND in Section 7.5, we shall first describe the basic concept of the algorithm and discuss some basic features.

7.3.1 Basic Concept

Our proposed topology and energy control algorithm (TECA) is motivated by a classic clustering approach. Clustering the network means that each node is assigned to a *cluster* of nodes in which one master node acts as the *cluster head*. The cluster head is responsible for all its assigned nodes and might perform special application tasks, like handling data aggregation, controlling access to the medium, or providing routing-related functions. Once the network has been divided into several clusters, TECA will select some nodes to act as *bridges* between two or more clusters. In this way, the entire network gets connected.

As shown in Figure 7.1, a sensor node can be in one of five states: *initial*, *sleeping*, *passive*, *bridge*, or *cluster head*. After a node is powered on, it will be in the initialization state with its radio turned on until a timer T_i expires. In this state, nodes overhear packet transmissions, build a neighborhood table, and measure link qualities to adjacent nodes. After time T_i has elapsed, a node will change its state to passive. Like in the initialization state, passive nodes overhear ongoing packet transmissions and keep their neighborhood tables up-to-date. Additionally, in the case of network disconnections, they will become *active*, either as a cluster head or a bridge. Otherwise, they will stay passive for a time T_p until they will go to sleep to save energy. We refer to a node as *sleeping* if it has turned its communication radio off. Other energy-consuming components like sensing and processing units may still be turned on.

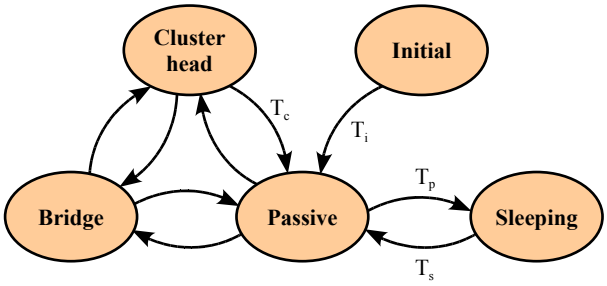


Figure 7.1: TECA's state transitions

Since TECA conserves energy by putting redundant nodes to sleep, a major challenge is to maintain connectivity in the network. Of course, a node with information about all nodes and the links between them has the ability to build a well-connected topology. However, TECA should be *self-configuring* and should work in a distributed and localized fashion. Therefore, after clustering the entire network, TECA maintains connectivity by selecting *bridges* that connect different clusters. We will describe the cluster head and bridge selection process in detail in the next section. In addition to maintaining network connectivity, TECA tries to select as few nodes as possible. Furthermore, it explicitly considers link qualities by measuring the links' packet delivery ratios, as packet losses are very common due to low-power radios, reflections, attenuation, and multipath/fading effects.

7.3.2 TECA in Detail

The topology built by TECA is based on neighborhood information that nodes exchange periodically. These *beacons* will be broadcast by non-sleeping nodes each time an announcement timer T_a expires. Beacons contain the node's id, state, residual energy, a timeout value, and 1-hop neighborhood information. However, only information about active neighbors, i. e., cluster heads and bridges, are included in the packet. To identify asymmetric links and to provide other nodes with link qualities, link loss information for each active neighbor is added. This information is based on local measurements and is adapted by exponential smoothing over time. Since packet loss may be different depending on the transmission direction, both directions are considered independently. Thus, we can identify asymmetric links through a difference in packet loss that exceeds a predefined threshold.

While a node is in the initialization state, it will only sends beacons every announcement time. After it has changed its state to passive, it will first set a passive timer T_p, after which it will turn its communication radio off and sleep to save energy. As long as a node is not sleeping, it will check its current state each time it receives or must send an announcement packet. The corresponding *CheckState(n)* function is shown in Algorithm 7.1.

Algorithm 7.1 $CheckState(Node\ n)$

```
1:  if IsCluster(n) then
2:      if n.state ≠ clusterhead then
3:          n.state ← clusterhead
4:          set cluster head timer T_c
5:      end if
6:  else if IsBridge(n) then
7:      if nstate ≠ bridge then
8:          n.state ← bridge
9:      end if
10: else
11:     if n.state ≠ passive then
12:         n.state ← passive
13:         set passive timer T_p
14:     end if
15: end if
```

Once a node has lost its cluster head and decided to be a cluster head itself, function *IsCluster(n)* will return true. Otherwise, the node will verify that it should be active to connect different clusters based on its 2-hop neighborhood information. If function *IsBridge(n)* also returns false, the node will change to (or stay in) the passive mode. We now consider both functions in more detail in the next two sections and describe how the cluster as well as the bridge selection process is performed.

Cluster Head Selection

The clustering selection process works as follows: As long as a node is not assigned to a cluster, it is a potential cluster candidate itself, which the node will propagate to its neighborhood. By using a predefined performance metric, e. g., residual energy, the best *suitable* node can be found, i. e., the node with the best cluster selection value. Eventually, that node will become a cluster head. All other nodes in the cluster head's 1-hop neighborhood will be assigned to it, and will no longer be cluster head candidates. With that mechanism, any node will finally either be a cluster head itself or be assigned to one.

Figure 7.2 depicts a possible cluster formation for a sample sensor network with five cluster heads. Each cluster is indicated by a circle around the cluster head that is equal to its radio transmission range. Although some nodes are within transmission range of more than one cluster head, they are assigned to just one cluster. Later, these nodes might become potential bridges (or bridge candidates) to connect two or more clusters with each other.

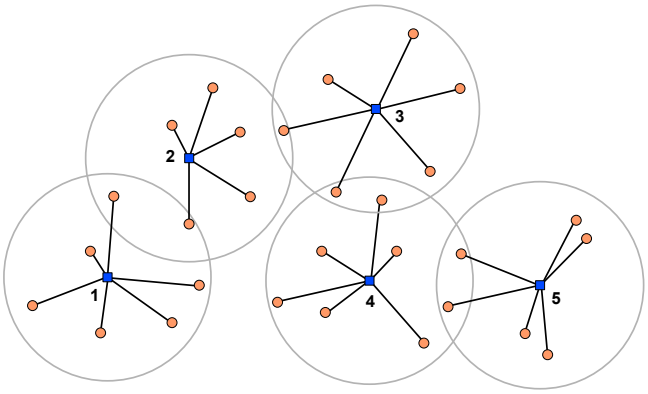
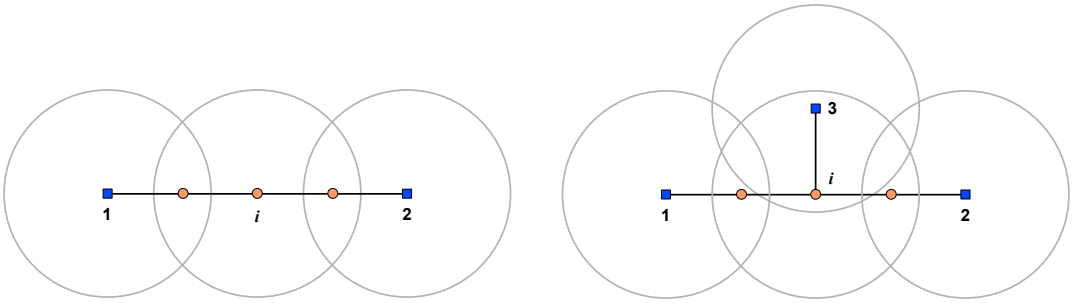


Figure 7.2: TECA's cluster formation

After the cluster selection process, *adjacent* cluster heads will be at most three hops apart, i. e., starting from an arbitrary cluster head, another one can be reached in at most three hops.

PROOF. Assume that there are more than two nodes between a cluster head 1 and another cluster head 2 as shown in Figure 7.3(a). Consider the node i that is two hops away from cluster head 1. Since this node is not a cluster head itself (otherwise, node 1 would reach another cluster head in fewer than three hops), it must be assigned to an additional cluster 3 that must be within its 1-hop neighborhood, as shown in Figure 7.3(b). Thus, starting from cluster head 1, another cluster head can be reached over node i in at most three hops. □



(a) (b)

Figure 7.3: Illustration of the number of hops between adjacent clusters

After a node has been selected as a cluster head, it will set a cluster timer T_c, during which it will not change its state. Due to load and energy balancing, it will try to find another cluster it could join to if T_c expires. The running time of T_c is defined as

$$T_c = \min\{\alpha, l_i\} \cdot L_i^{init}, \tag{7.1}$$

with $\alpha \in [0 \dots 1]$ being the *cluster timeout factor*, L_i^{init} being the initially assigned energy in time units, and l_i being the fraction of node i's residual energy (lifetime).

The algorithm of the cluster head selection is shown in Algorithm 7.2. First, the best cluster head with respect to residual energy is determined from among all 1-hop neighbors. Note that we only consider neighbors with a packet loss rate below a predefined *loss threshold* LT. In the case of indecision

regarding the nodes' energy values, we use the node's id as a tie breaker. If no existing cluster head is found, the node will become a cluster head itself. However, if the node is already a cluster head and has found another one whose cluster timeout timer has not expired and whose residual energy is higher, it will change its state and become part of the newly-found cluster.

Algorithm 7.2 $IsCluster(Node\ n)$

```
1:  if n.state = clusterhead ∧ time < n.T_c then
2:      return true
3:  end if
4:  c ← null
5:  for all Neighbors m ∈ N : 1 − prr_{n,m} · prr_{m,n} ≥ LT do
6:      if m.state = clusterhead ∧ m.T_c < time then
7:          if !c ∨ m.energy > c.energy ∨ (m.energy = c.energy ∧ m.id < c.id) then
8:              c ← m
9:          end if
10:     end if
11: end for
12: if !c ∨ (n.state = clusterhead ∧ (n.energy > c.energy ∨
                                      (n.energy = c.energy ∧ n.id < c.id))) then
13:     c ← n
14: end if
15: return c.id = n.id
```

Bridge Selection

The next step after decomposing the entire network into clusters is to select bridge nodes to connect clusters to each other. Other nodes that are neither cluster heads nor bridges will turn their radios off and sleep without participating in any network communication. However, the cluster heads to which sleeping nodes have been assigned are responsible for them. For example, cluster heads may store and forward data packets to sleeping nodes as soon as these wake up. If sleeping nodes have detected an event that should be forwarded, e. g., to a network sink, they can wake up immediately and send their data to the cluster head, which forwards the data on to the sink. As every node is assigned to a cluster head, which in turn maintains connectivity to the sink, local events can thus be propagated and processed at any time.

After the cluster head selection process, all non-cluster heads will remain passive until their passive timers T_p expire. During this phase, they will listen to announcement packets for their neighbors. If a passive node is aware of the existence of more than one cluster, that node will become a bridge candidate.

Determining which of these nodes will become bridges is a great challenge. There are three main requirements that should be taken into account: Bridges must connect different clusters in an optimal way, i. e.,

1. the packet reception ratio between clusters should be maximized,

2. the connection should be long-lived, and

3. the number of selected bridges should be minimal to prolong the entire network's operational lifetime.

The basic idea of our bridge selection algorithm is to consider a node's 2-hop neighborhood as a graph with different link costs. Then, we compute the *minimum spanning tree* (MST) of the graph, but only take *virtual links* between cluster heads into account. Virtual links are composed of one or maybe several nodes that connect cluster heads in the best way with respect to the above requirements. Thus, a virtual link is the best path for connecting two cluster head with each other. Figure 7.4 shows an example in which all five cluster heads are connected by such virtual links.

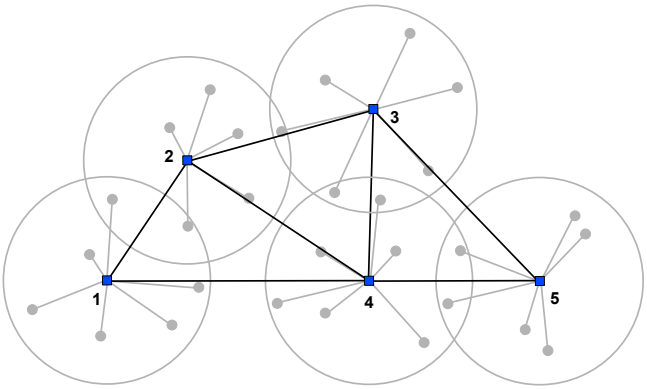


Figure 7.4: Virtual cluster links in TECA

To reflect both the link's packet reception ratio and its lifetime, link costs are introduced. Later we will additionally use a *penalty cost* to minimize the number of active nodes. The packet reception ratios of virtual links are computed as follows: Given a virtual link containing k nodes $n_1 \dots n_k$, with n_1 and n_k being cluster heads, let prr_i, $1 \leq i \leq k$, be the packet reception ratio between n_1 and n_i and $prr_{i-1,i}$, respectively $prr_{i,i-1}$ the reception ratio between two successive nodes. We then define prr_i as

$$prr_i = \begin{cases} 1 & i = 1, \\ \prod_{j=2}^{i} prr_{j-1,j} \cdot prr_{j,j-1} & i = 2 \dots k. \end{cases} \tag{7.2}$$

Thus, the virtual link's packet reception ratio is defined as prr_k[2].

Also, the lifetime of a virtual link is defined as $lifetime_k$ with

$$lifetime_i = \begin{cases} l_i & i = 1, \\ \min\{lifetime_{i-1}, l_i\} & i = 2 \dots k, \end{cases} \tag{7.3}$$

where l_i is the node's residual energy fraction.

[2]Note that prr_k is defined as the product of the links' packet reception ratios regarding both directions in order to reflect successful transmissions of data packets and acknowledgements.

Then, the cost of a virtual link is defined as a priority function f that combines the link lifetime and packet reception ratio as

$$c_i = 1 - f(prr_i, lifetime_i). \tag{7.4}$$

We will investigate this priority function in the next section in more detail.

Figure 7.5 first shows one possible mapping between virtual links and activated nodes - the bridges. Of course, more bridges than necessary have been selected than if we had computed the global MST on the entire network. However, note that all nodes just have a localized view of the network and thus are only able to take 2-hop neighborhood information into account.

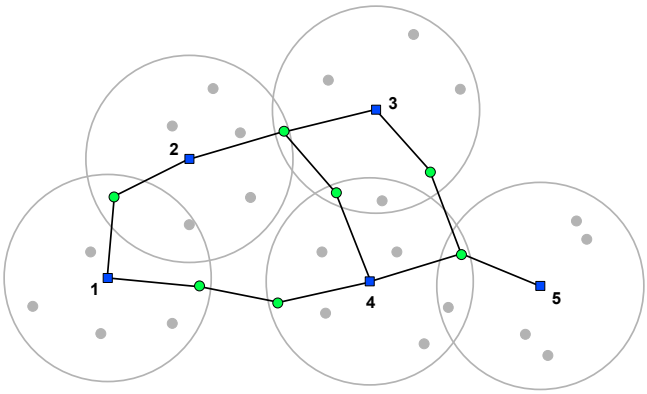


Figure 7.5: Built topology

The appropriate bridge selection algorithm is shown in Algorithms 7.3 and 7.4. First, let us have a look at the $IsBridge(n)$ function. If a node n already acts as a cluster head with its cluster timer T_c still running, the node will not change its state. Furthermore, if the node has discovered fewer than two clusters in its 2-hop neighborhood N^2, it will not have to be active. Otherwise, using function $BuildTopology(n)$, the node will build a *minimum cluster spanning tree* (MCST) that consists of all cluster heads and bridges known. The set $MCST^N$ will then contain all neighbors that *should* be active in order to build the backbone topology, i. e., cluster heads and bridge nodes. Thus, $IsBridge(n)$ will return true if $MCST^N$ contains node n.

The MCST built in Algorithm 7.4 relies on a priority search on the neighborhood graph [63]. Nodes that are dead, i. e., without any residual energy, or are asleep are not considered. The crucial point of the algorithm is that the MST is not constructed with respect to all nodes but just to cluster heads, i. e., by considering virtual links only. In order to obtain the MST, we employ a heap, which implements a priority queue to find the next node with a minimum cost, starting at an arbitrary cluster head. The node's cost, as a combination of packet reception ratio and lifetime, is computed according to Equation 7.2, 7.3, and 7.4.

The virtual links are then established by using the set $virtual\_link$ of node n. Each time a new cluster head is visited, all nodes along the virtual link are added to $MCST^N$. In addition, the appropriate node is marked (with cost equal to $-\infty$) and its reception ratio, lifetime, and virtual link set are reset.

Algorithm 7.3 $IsBridge(Node\ n)$

if $n.state = clusterhead\ \vee\ |\{Nodes\ m \in N^2 : m.state = clusterhead\}| < 2$ **then**
  return false
end if
$MCST^N \leftarrow BuildTopology(n)$
return $n \in MCST^N$

Algorithm 7.4 $BuildTopology(Node\ n)$

for all $Nodes\ m \in N^2 \cup \{n\}$ **do**
  $m.cost \leftarrow (m.state \in \{sleeping, dead\})\ ?\ -\infty : \infty$
  $m.virtual\_link \leftarrow \{m\}$
end for
$MCST^N \leftarrow \emptyset$
for all $Nodes\ m \in N^2 : m.state = clusterhead$ **do**
  if $m.cost \neq -\infty$ **then**
    $m.lifetime \leftarrow m.energy$
    $m.prr \leftarrow 1$
    $MCST^N \leftarrow MCST^N \cup \{m\}$
    $queue.push(m, -\infty)$
    repeat
      $v \leftarrow queue.pop()$
      if $v.state = clusterhead\ \wedge\ v.cost \neq -\infty$ **then**
        $MCST^N \leftarrow MCST^N \cup v.virtual\_link$
        $v.lifetime \leftarrow v.energy$
        $v.prr \leftarrow 1$
        $v.cost \leftarrow -\infty$
        $v.virtual\_link \leftarrow \{v\}$
      end if
      for all $Neighbors\ w \in N_v^1 : w \notin v.virtual\_link$ **do**
        if $w.cost \neq -\infty$ **then**
          $l \leftarrow \min\{v.lifetime, w.energy\}$
          $prr \leftarrow v.prr \cdot prr_{v,w} \cdot prr_{w,v}$
          $prr \leftarrow (1 - prr < LT)\ ?\ \varepsilon : prr$
          $c \leftarrow 1 - f(prr, l)$
          if $c < w.cost\ \vee\ (c = w.cost\ \wedge\ Cmp(v.virtual\_link \cup \{w\}, w.virtual\_link) < 0)$ **then**
            $w.lifetime \leftarrow l$
            $w.prr \leftarrow prr$
            $w.cost \leftarrow c$
            $w.virtual\_link \leftarrow v.virtual\_link \cup \{w\}$
            $queue.update(w, c)$
          end if
        end if
      end for
    until $!queue.empty()$
  end if
end for
return $MCST^N$

After visiting node v, its 1-hop neighborhood N^1 is processed. However, nodes that are already contained in v's virtual link set are skipped to avoid loops. For all other nodes, the nodes' costs are computed according to Equation 7.4.

In order to get a well-connected topology, the reception ratio of a virtual link will artificially decrease if its loss rate is higher than LT. In this case, the reception ratio will be set to a predefined value ε with $0 < \varepsilon \ll 1$. In so doing, other virtual links may be preferred even if their costs (possibly influenced by the lifetime) may actually be worse.

If an adjacent node w is reachable over node v with lower cost, w's variables $lifetime$, prr, $virtual\_link$, and $cost$ will be updated. Additionally, the heap will be updated. Should both costs be the same, the virtual link will be used as a tie-breaker. Therefore, function $Cmp(link_1, link_2)$ compares two virtual links, returning minus one if the former is better, i. e., if

1. $|link_1| < |link_2|$, or
2. $|\{n \in link_1 : n.state = \{clusterhead, bridge\}\}| > |\{m \in link_2 : m.state = \{clusterhead, bridge\}\}|$, or
3. $\sum_{n \in link_1} n.energy > \sum_{m \in link_2} m.energy$, or
4. $\min_{n \in link_1 \wedge n \notin link_2}\{n.id\} < \min_{m \notin link_1 \wedge m \in link_2}\{m.id\}$.

Note that the node's cost mainly depends on the priority function f, which we consider in the next section in more detail.

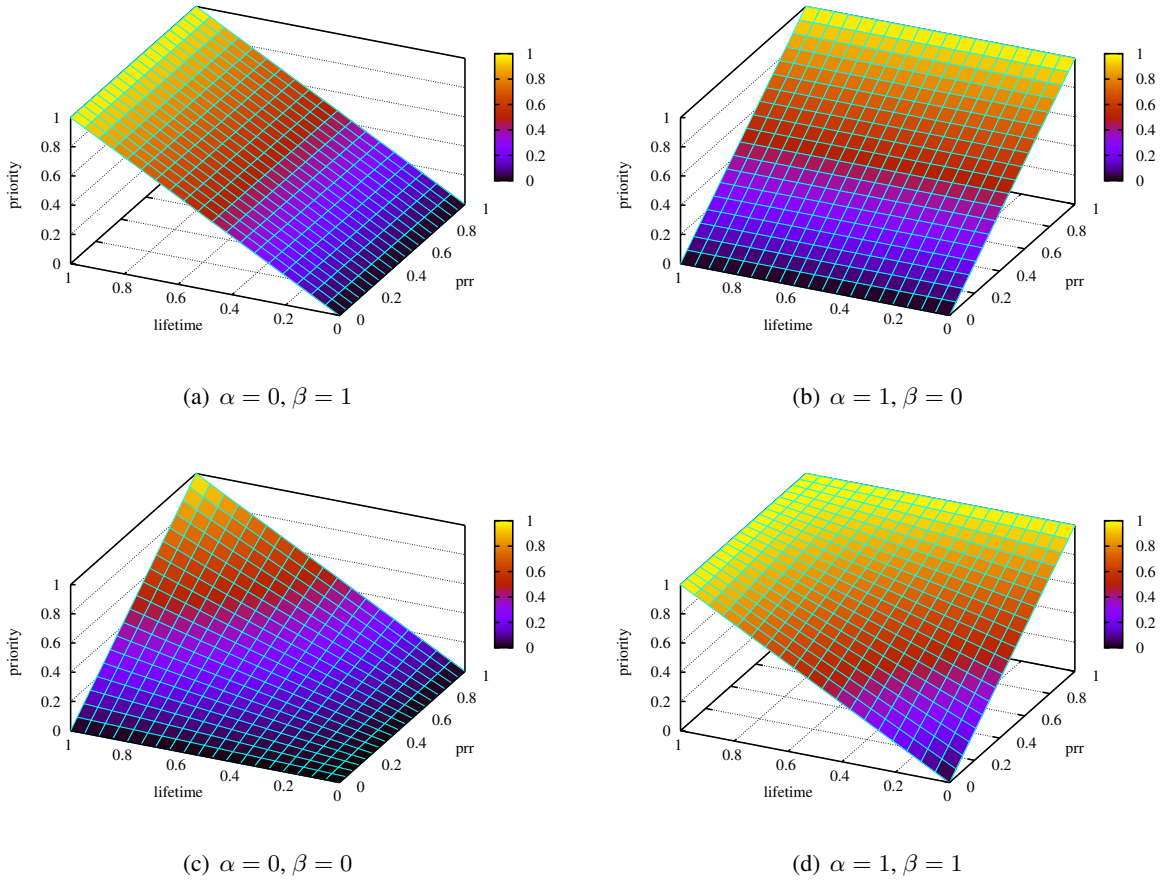
PRR Lifetime Model

The priority function f uses two parameters, the packet reception ratio and a lifetime factor. Based on both values, it determines the node's priority expressed as a value within $[0..1]$. The higher the priority, the higher the probability to visit the appropriate node next. Certainly, there are many possible functions that would fulfill this requirement. We propose a two-dimensional linear function controlled by two parameters α and β, which is defined as

$$f(prr, lifetime) = prr\,[(1-\alpha) \cdot lifetime + \alpha] + (1 - prr) \cdot \beta \cdot lifetime. \tag{7.5}$$

In Figure 7.6, the priority function is plotted for some α, β pairs. Changing the parameters α and β influenced the weighting of prr and $lifetime$. For example, a topology just based on residual energy will be achieved by using $\alpha = 0$, $\beta = 1$. On the other hand, $\alpha = 1$, $\beta = 0$ will lead to a topology optimized in terms of packet loss. Note that these cases can also be achieved by using a simpler linear combination like $\gamma \cdot prr + (1-\gamma) \cdot lifetime$, respectively by using $\alpha = \gamma$ and $\beta = 1 - \gamma$. However, satisfying the following requirements would then not be possible:

1. Nodes with a reception ratio close to zero as well as



(a) $\alpha = 0, \beta = 1$

(b) $\alpha = 1, \beta = 0$

(c) $\alpha = 0, \beta = 0$

(d) $\alpha = 1, \beta = 1$

Figure 7.6: Priority function f with different α, β values

2. nodes with little residual energy should be considered worthless[3].

But if we set $\alpha = 0$, $\beta = 0$, the priority function will exactly express the desired behavior, as shown in Figure 7.6(c). In Section 7.4, we will later investigate the influence of different α, β values by means of simulations and evaluate how the performance of TECA is affected.

Penalty Cost

As described above, the third requirement of the bridge selection process is to minimize the number of selected bridges to save as much energy as possible. For example, consider the sub-graph depicted in Figure 7.7. If we take only the link loss (depicted as weights on each link) into account and neglect the link lifetime, all nodes 6, 7, and 8 will be selected as bridges. The MCST will then contain link $(2, 7, 3)$ with cost (packet loss) 0.0 and link $(2, 6, 8, 4)$ with cost 0.1. However, if we also take the energy consumption into account and accept higher link costs, link $(2, 7, 8, 4)$ could be an alternative since node 6 then could sleep. Thus, the main question is how to tackle both cases.

[3]In this context, the term "worthless" means that the node's priority should be zero such that other nodes will be preferred regarding the constructed topology.

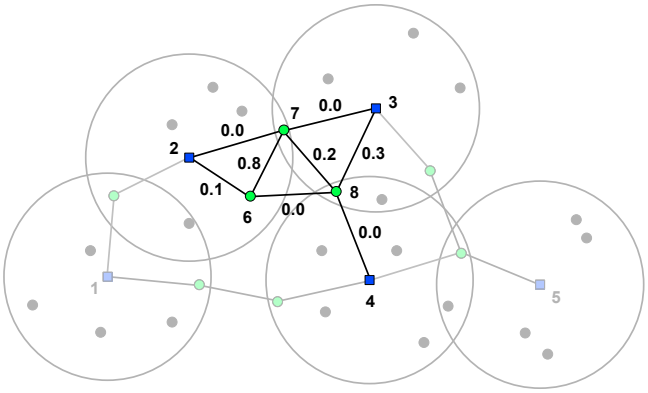


Figure 7.7: Topology showing different link costs

We will handle this situation by introducing a penalty cost. The idea is to penalize all nodes that are not yet part of the backbone topology. For example, consider Figure 7.7 again. Assuming the bridge selection algorithm shown in Algorithm 7.4 starts in node 2, link $(2, 7, 3)$ will be the first virtual link added to MCST because all nodes 6, 7, and 8 will cause penalty costs. Next, node 7 will become (or remain) active and will later be preferred over other nodes, i. e., it will no longer be penalized. Then, depending on the used penalty value PV, link $(2, 7, 8, 4)$ might have a lower cost than link $(2, 6, 8, 4)$, despite a lower packet reception ratio.

We compute the penalty cost of a virtual link as follows: Given a virtual link containing k nodes $n_1 \dots n_k$, with n_1 and n_k being cluster heads, let $penalty_i$ be assigned to node n_i with $1 \leq i \leq k$. With PV denoting the *penalty value* ($0 \leq PV < 1$) and specifying $0 < \varepsilon \ll 1$, the penalty cost on the link from node n_1 to n_i is defined as

$$penalty_i = \begin{cases} \rho_i & i = 1, \\ penalty_{i-1} + (1 - penalty_{i-1}) \cdot \rho_i & i = 2 \dots k, \end{cases} \tag{7.6}$$

with

$$\rho_i = \begin{cases} \varepsilon + PV & n_i \neq \{clusterhead, bridge\} \ \vee (n_i = \{clusterhead, bridge\} \wedge n_i \notin MCST^N), \\ \varepsilon & \text{otherwise.} \end{cases}$$

Then, the cost function defined in Equation 7.4 changes to

$$c_i = \omega_i + (1 - \omega_i) \cdot penalty_i, \tag{7.7}$$

with $\omega_i = 1 - f(lifetime_i, loss_i)$ and $i = 2 \dots k$.

The constant ε is used to penalize longer links in terms of hops. Furthermore, passive nodes will always receive penalty costs until they become active. For example, consider node 6 in Figure 7.7. If PV is high enough that link $(2, 7, 8, 4)$ has been added to MCST, node 6 will become a sleeping node. However, in this case, another problem may occur. If node 7 is aware of another link outside the scope of node 6 which connects cluster heads 2 and 3 better than link $(2, 7, 3)$, it may not become active at

all if it chooses link $(2, 6, 8, 4)$ to connect clusters 2 and 4. Consequently, node 6, as well as node 7, goes to sleep.

Hence, just adding a penalty cost to Algorithm 7.4 could lead to disconnected clusters depending on the *order* in which nodes are added to MCST, or more precisely depending on the order in which already activated nodes like cluster heads and bridges are added to MCST. Since the topology is computed in a distributed fashion, each node likely builds the MCST on different sub-graphs representing the node's 2-hop neighborhood. Thus, we cannot guarantee that for each node Algorithm 7.4 will start with the same cluster head, which might lead to different MCSTs.

For example, consider Figure 7.7 again. Let PV be 0.2, and the amount of residual energy be negligible. Furthermore, let us assume that nodes 6, 7, and 8 are bridges, and that the bridge selection algorithm of node 6 starts at cluster head 2, and that of node 7 at cluster head 4. First, let us consider the MCST computed by node 6. Because link $(2, 7, 3)$ has minimal cost, it will be added to node 6's MCST first. As node 7 should be a bridge, it will not receive a penalty cost. Thus, link $(3, 7, 8, 4)$ is better than link $(2, 6, 8, 4)$, and it is added next. That terminates the algorithm, with node 6 becoming passive.

The MCST of node 7 is built as follows: Since the bridge selection algorithm starts at node 4, link $(4, 8, 6, 2)$ will be added first. However, the next best link will then be $(4, 8, 3)$. Thus, node 7 will assume that it is redundant and become passive. Hence, in the end, both nodes 6 and 7 will be passive, likely resulting in a partitioned network.

As we have seen, the graph's traversal order is crucial and may lead to different MCSTs if link/node cost changes during the traversal. Therefore, we propose the following algorithmic approach:

1. Search for the best virtual link in the graph, i. e., the link with the minimum cost.
2. Add that link to MCST.
3. Now search for the next best virtual link in the graph that is not yet contained in MCST.
4. If such a link exists, go back to step 2. Otherwise, terminate.

In this way, we can enhance Algorithm 7.4 by using the cost function defined in Equation 7.7. The extended bridge selection algorithm is shown in Algorithms 7.5 and 7.6. Set $MCST^L$ contains the selected MCST's links, whereas $MCST^N$ contains the selected nodes. Then, the MCST is built up successively. Virtual links are added to $MCST^L$ according to their link costs, starting with the best link. The algorithm will terminate if no further link can be found, i. e., the MCST is complete. The number of times the priority search function will be called is bounded by $M - 1$, where M denotes the number of cluster heads, because $MCST^L$ contains at most $M - 1$ virtual links.

Referring to the last example, now both nodes 6 and 7 will add link $(2, 7, 3)$ to MCST first, independent of the traversal's starting point. Afterwards, link $(2, 7, 8, 4)$ will be added next, as node 7 has already been selected and thus does not receive a penalty cost. Consequently, only node 6 will become passive, while node 7 will stay in its bridge state.

Algorithm 7.5 $BuildTopology(Node\ n)$

1: $MCST^N \leftarrow \{Nodes\ m \in N^2 : m.state = clusterhead\}$
2: $MCST^L \leftarrow \emptyset$
3: **repeat**
4: $best\_link \leftarrow PrioritySearch(n, MCST^N, MCST^L)$
5: **if** $best\_link \neq \emptyset$ **then**
6: $MCST^N \leftarrow MCST^N \cup best\_link$
7: $MCST^L \leftarrow MCST^L \cup \{best\_link\}$
8: **end if**
9: **until** $best\_link = \emptyset$
10: **return** $MCST^N$

Sleeping Timeout

After cluster heads and bridge nodes have been selected, all remaining nodes will stay passive until their passive timers T_p have expired. The intuition behind passive nodes is the ability to react to changes in the neighborhood before nodes go to sleep. For example, until the topology has reached a stable state, nodes that have already been activated might become passive again, requiring other nodes to be active instead.

If T_p expires, a passive node will go to sleep and will wake up (at last) at the cluster timeout to participate in rebuilding the topology. However, if only the timeout of the assigned cluster head is considered, network partitions may result. For example, Figure 7.8 shows a topology of five nodes. Let nodes 1 and 3 be cluster heads, node 2 be a bridge, and nodes 4 and 5 be sleeping nodes. Furthermore, let node 4 be assigned to cluster 1, and nodes 2 and 5 be assigned to cluster 3. Assuming that the timeout of cluster 3 is before that of cluster 1, node 5 will wake up first but will not find a cluster to join. Thus, it will become a cluster head itself. If node 4 does not wake up at the same time, the network will be partitioned since node 5 will not be connected to nodes 1 and 2. Moreover, if node 2 dies due to lack of energy before the timeouts of clusters 1 and 3, both clusters will be partitioned, too. In such a case, node 4 must wake up in time to take over the role of node 2.

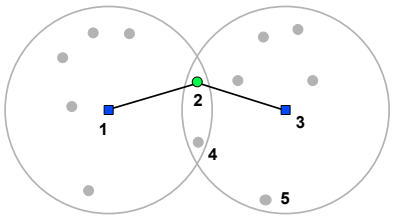


Figure 7.8: Sleeping timeout example

Hence, a node will wake up if

1. a cluster timeout of a known cluster in the node's neighborhood has occurred, or
2. a bridge has run out of energy, with the network remaining connected if the considered node will be active.

Algorithm 7.6 $PrioritySearch(Node\ n, MCST^N, MCST^L)$

```
 1: for all Nodes m ∈ N^2 ∪ {n} do
 2:     m.cost ← (m.state ∈ {sleeping, dead}) ? −∞ : ∞
 3:     m.virtual_link ← {m}
 4: end for
 5: best_link ← ∅
 6: best_cost ← ∞
 7: for all Nodes m ∈ N^2 : m.state = clusterhead do
 8:     if m.cost ≠ −∞ then
 9:         m.lifetime ← m.energy
10:         m.prr ← 1
11:         queue.push(m, −∞)
12:         repeat
13:             v ← queue.pop()
14:             if v.state = clusterhead ∧ (v.cost ≠ −∞) then
15:                 if v.virtual_link ∉ MCST^L ∧ (v.cost < best_cost ∨
                      (v.cost = best_cost ∧ Cmp(v.virtual_link, best_link) < 0)) then
16:                     best_link ← v.virtual_link
17:                     best_cost ← v.cost
18:                 end if
19:                 v.lifetime ← v.energy; v.prr ← 1; v.cost ← −∞
20:                 v.virtual_link ← {v}
21:             end if
22:             for all Neighbors w ∈ N_v^1 : w ∉ v.virtual_link do
23:                 if w.cost ≠ −∞ then
24:                     l ← min{v.lifetime, w.energy}
25:                     prr ← v.prr · prr_{v,w} · prr_{w,v}
26:                     prr ← (1 − prr > LT) ? ε : prr
27:                     p ← ε
28:                     if w.state ∉ {clusterhead, bridge} ∨
                          (w.state ∈ {clusterhead, bridge} ∧ w ∉ MCST^N) then
29:                         p ← p + PV
30:                     end if
31:                     p ← v.penalty + (1 − v.penalty) · p
32:                     c ← (1 − f(prr, l)) + f(prr, l) · p
33:                     if c < w.cost ∨ (c = w.cost ∧
                                Cmp(v.virtual_link ∪ {w}, w.virtual_link) < 0) then
34:                         w.lifetime ← l
35:                         w.prr ← prr
36:                         w.penalty ← p
37:                         w.cost ← c
38:                         w.virtual_link ← v.virtual_link ∪ {w}
39:                         queue.update(w, c)
40:                     end if
41:                 end if
42:             end for
43:         until !queue.empty()
44:     end if
45: end for
46: return best_link
```

Based on the MCST, the sleeping timeout of a node n is calculated as follows: First, the minimum of all known cluster timeouts is determined. Then, Algorithm 7.5 is used to identify *superior* nodes that "suppress" node n to keep it from becoming active. E. g., in Figure 7.8, node 2 is a node superior to node 4. Therefore, Algorithm 7.5 needs to be modified as follows: Let Ω be the set of active nodes (cluster heads and bridges) in the node n's 2-hop neighborhood. Unlike before, the priority search is performed on $\Omega \cup \{n\} \backslash \{i\}$ for each bridge i. If for this case node n needs to be active, bridge i will a node superior to n. Additionally, let Ω_C be the set of known cluster heads, and Ω_S be the set of nodes superior to n. The sleeping timeout is then calculated as

$$T_s = \min_{j \in \Omega_C \cup \Omega_S} \{t_j\}, \tag{7.8}$$

with t_j being the cluster timeout if $j \in \Omega_C$. If $j \in \Omega_S$, the timeout will be determined by the node's residual energy value.

7.4 Performance Evaluation of TECA

As described in the last section, TECA's performance relies on the user-defined parameters α, β, and PV, which have a significant impact on the topology being built. While α specifies the fraction with which packet reception ratios on links are considered, β affects the link's lifetime in the final topology. PV controls the penalty cost fraction, which is used to keep the number of activated nodes as low as possible, and thus maximizes the number of sleeping nodes. In the following, we investigate the influence of these parameters by means of simulation.

7.4.1 Simulation Setup

We use the following simulation setup: Nodes are placed randomly in an area of size 100×100 m^2 by using a uniform distribution function. All simulations are based on static networks with a node density μ of 20, specifying the average number of neighbors within a node's radio transmission range. For each pair of nodes, we computed the packet reception ratio based on the model presented in Chapter 4. The loss threshold NT is set to 0.8. According to the appropriate link loss rate, packets are dropped randomly in the network. Packet collisions are not taken into account since in the majority of cases they depend on the employed MAC schemes, which are beyond the scope of this chapter.

In order to highlight only the influence of different values for α, β, and PV, we assume that all nodes know their neighbors and have sufficient information about their packet reception ratios. As β can only be evaluated for different link lifetimes, the node's residual energy faction will be randomly chosen within $(0..1]$. Then, for all parameter combinations, the TECA algorithm is started on each node. The simulation continues until no more state changes occur, and the built topology is stable. In total, we carried out 200 simulation runs.

7.4.2 Performance Metrics

In order to investigate the performance of TECA based on different parameter settings, we are interested in the following metrics: the *mean cluster link lifetime* ($MCLL$), the *mean cluster link loss rate* ($MCLLR$), and the *mean number of active nodes* ($MNAN$). $MCLL$ and $MCLLR$ are computed by taking all virtual links of the *global* MCST into account, which is built according to Algorithm 7.4, but with two modifications: (i) Only activated nodes, i. e., cluster heads and bridge nodes, are considered, and (ii) neighbor tables of all nodes are taken into account simultaneously in order to get a global spanning tree.

$MCLL$ then indicates the average lifetime until the MCST gets partitioned. Similarly, $MCLLR$ specifies the quality with which cluster heads are connected. The formal definitions of $MCLL$, $MCLLR$, and $MNAN$ are as follows: Assuming there are P partitions in the global topology of cluster heads and bridges, let $MCST_p$ be the MCST of partition p, C_p be the number of cluster heads, and B_p be the number of selected bridges in partition p. The number of cluster links per partition is equal to $C_p - 1$ due to the spanning tree property of $MCST_p$. Furthermore, let n_{ijp} be the i-th node of a cluster link j in partition p, with $energy_{ijp}$ being the node's residual energy fraction, and l_{jp} being the length of link j in hops.

Then, $MCLL$ is defined as

$$MCLL = \frac{\sum_{p=1}^{P} \sum_{j=1}^{C_p-1} \min_{1 \le i \le l_{jp}} \{energy_{ijp}\}}{\sum_{p=1}^{P} (C_p - 1)}. \tag{7.9}$$

According to Equation 7.2, let $loss_{jp}$ be the loss rate of cluster link j containing l_{jp} nodes in partition p, i. e., with $loss = 1 - prr$. Thus, $MCLLR$ can be calculated from

$$MCLLR = \frac{\sum_{p=1}^{P} \sum_{j=1}^{C_p-1} loss_{jp}}{\sum_{p=1}^{P} (C_p - 1)}. \tag{7.10}$$

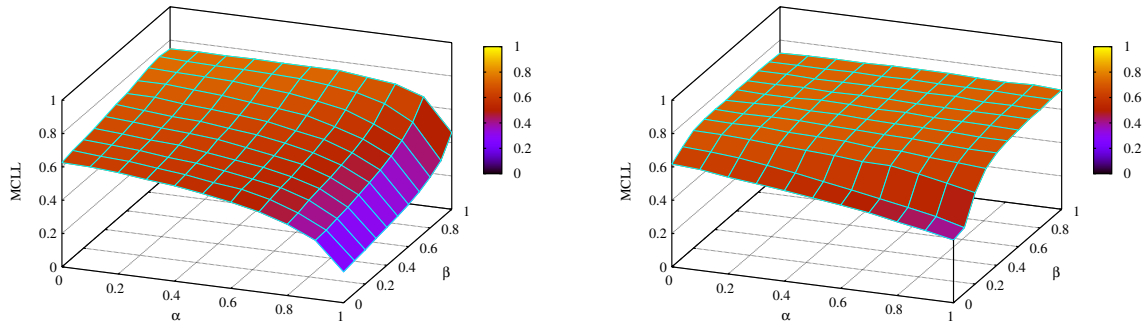
For $MNAN$, we get

$$MNAN = \frac{\sum_{p=1}^{P} (C_p + B_p)}{N}, \tag{7.11}$$

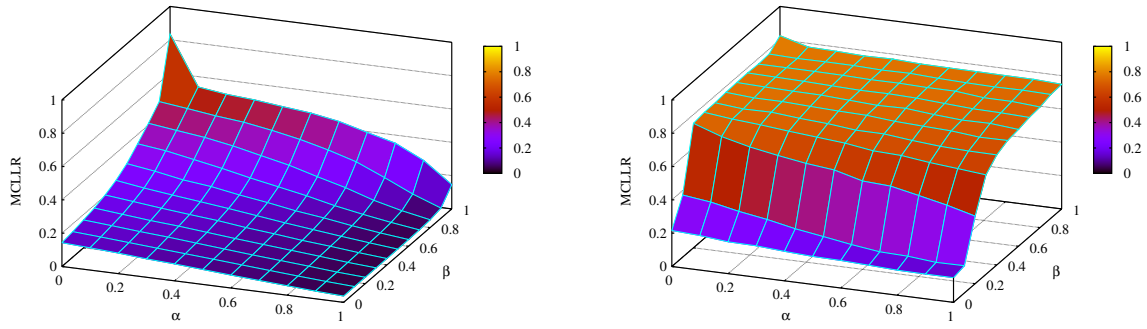
with N being equal to the total number of network nodes.

7.4.3 Simulation Results

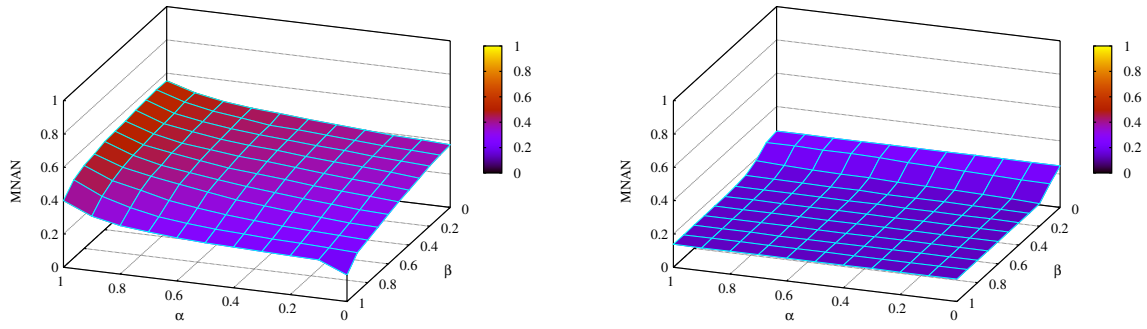
Figure 7.9 depicts the simulation results for $PV = 0$ and $PV = 0.8$. Starting with the results for $PV = 0$, Figure 7.9(a) shows the average cluster link lifetime $MCLL$. As expected, the highest value is achieved if just lifetime is taken into account (by TECA's priority function f) and packet reception ratios are not considered, which is expressed as $\alpha = 0$, $\beta = 1$. On the other hand, if the link lifetime is neglected completely ($\alpha = 1$, $\beta = 0$), $MCLL$ is worst. However, for $\alpha < 1$, nodes having



(a) $MCLL\,(PV = 0)$ (b) $MCLL\,(PV = 0.8)$



(c) $MCLLR\,(PV = 0)$ (d) $MCLLR\,(PV = 0.8)$



(e) $MNAN\,(PV = 0)$ (f) $MNAN\,(PV = 0.8)$

Figure 7.9: Simulation results for different α, β, and PV values ($\mu = 20$)

high reception ratios are prioritized in terms of their residual energy levels, without using node ids as tie-breakers[4]. Hence, the expected $MCLL$ increases.

The average cluster link loss rate $MCLLR$ depicted in Figure 7.9(c) shows a low loss rate for most α-β-pairs, with $\alpha = 1$ and $\beta = 0$ performing best. As activating additional bridge nodes causes no penalty cost, the $MCLLR$ is small as long as $\alpha \gg 0$ and $\beta \ll 1$. If only lifetime is taken

[4]See function $Cmp(link_1, link_2)$ in Section 7.3.2.

into account ($\alpha = 0$, $\beta = 1$), the resulting $MCLLR$ is worst. Because direct links between cluster heads usually have the longest link lifetimes (note that cluster heads are selected on the basis of their residual energies), the loss probability on such links is often very high, likely higher than the selected loss threshold LT. Hence, $MCLLR$ increases for $\beta \to 1$.

As shown in Figure 7.9(e), the average number of active nodes $MNAN$ (cluster heads and bridge nodes) varies between 18% and 55%. The best result is achieved for $\alpha = 0$ and $\beta = 1$. In this case, packet reception ratios are neglected, and the decision as to whether or not bridge nodes are activated depends solely on the lifetime of links that connect cluster heads to each other. Thus, direct links consisting of no intermediate bridges leads to a smaller $MNAN$. Since the lifetimes of direct links rely only on the energy of cluster heads, they are preferred over other links, even if the packet reception ratios between those clusters are worse than LT. Reducing the number of selected bridges is also feasible if the number of cluster heads connected by a single bridge node increases, i. e., a bridge is used to connect more than two cluster heads at the same time. However, the $MNAN$ cannot be minimized arbitrarily because TECA will always try to maintain connectivity in the first place.

Figures 7.9(b), 7.9(d), and 7.9(f) show the $MCLL$, $MCLLR$, and $MNAN$ if activating additional bridges cause a penalty cost of 0.8. As intended, $MNAN$ decreases, especially for $\beta \to 1$, which minimizes the influence of the packet reception ratio on the overall activation cost. As a consequence, if β increases, $MCLLR$ changes significantly for the following reason: According to Figure 7.6, direct links between cluster heads which have maximal lifetimes but minimal packet reception ratios experience costs of $1 - \beta$. Thus, for small β values, penalty costs are compensated by better delivery ratios due to more bridges. However, for $\beta \to 1$, using additional bridge nodes between cluster heads are avoided, unless direct links do not exist. This is also shown in Figures 7.9(e) and 7.9(f) by the $MNAN$ metric.

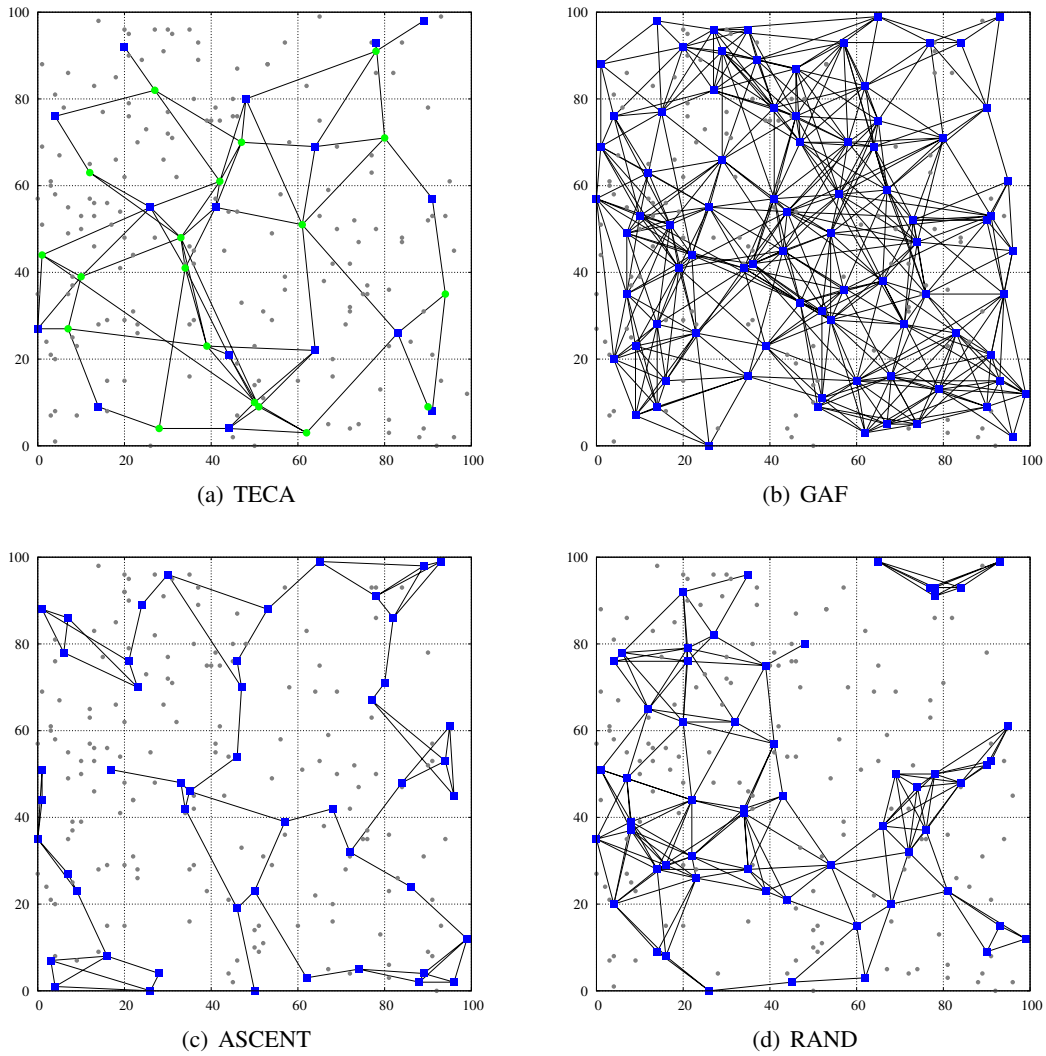
In conclusion, the simulation results show that setting the parameters right is a difficult and application-dependent task, influencing any decision to maximize either the time the topology remains stable or the packet delivery ratio between clusters. This trade-off can actually only be solved if the application's aim of the wireless sensor network is taken into account. We thus make the following assumptions: The packet delivery ratio within the network is assumed to be very important and is thus considered with high priority. In addition, link lifetimes of connected cluster heads should not be neglected in order to balance the energy consumption among all nodes evenly and achieve stable topologies. Especially at the end of the network's lifetime, it may be beneficial to use nodes with more residual energy because non-active nodes will then sleep longer. Hence, we set both α and β to zero. Furthermore, we set the the penalty cost to 0.8 to keep the number of active nodes as low as possible. Bridge nodes are thus forced to connect as many cluster heads as feasible, accounting for a maximum packet loss of 20% between them.

7.5 Simulative Comparison to other Approaches

We have conducted extensive simulations comparing TECA with other proposed approaches, namely GAF, ASCENT, and a randomized topology algorithm called RAND. While GAF [262] uses ge-

ographic information to build the topology, ASCENT [50] is based on a neighbor threshold NT, bounding the number of active nodes as described in Section 7.2. In contrast to TECA, neither of them takes the network connectivity into account explicitly. Preventing network partitions is addressed by a smaller grid size in GAF, a higher neighbor threshold in ASCENT, or a higher activation probability in RAND. However, connectivity is not guaranteed. Thus, applications might fail due to disconnections.

Before we compare those four algorithms in detail in different simulation scenarios, Figure 7.10 gives a first impression of how the topologies built of active nodes may look. For a density μ of 20 nodes, it is clearly shown that the topology of GAF consists of the most nodes, while TECA, ASCENT, and RAND use fewer active nodes. However, the topologies of ASCENT and RAND are already partitioned, although all sleeping nodes in ASCENT have five active neighbors as proposed in [50], and RAND activates nodes with a probability of 25%.



(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.10: Backbone topologies of TECA, GAF, ASCENT, and RAND ($\mu = 20$)

7.5.1 Simulation Setup

All algorithms have been simulated using the same setup as in the previous section, considering a network area of 100×100 m^2. In the following, we investigate the influence of the node density, the initial energy value, the cluster timeout factor, as well as the network performance over time. In so doing, we would like to answer the following questions:

1. How many nodes are selected by TECA as cluster heads, bridges, passive and sleeping nodes? How does the topology change over time? How many nodes do GAF, ASCENT, and RAND each select?

2. What is the energy consumption over time? How much can the network's operational lifetime be extended?

3. Are there still situations where network partitions occur? How loss-resistent is the topology in terms of the end-to-end packet delivery?

4. How does the network lifetime scale with respect to node density, initial energy, and the cluster timeout factor?

To obtain initial answers to these questions, we have focused the simulations on the node's selection process that is responsible for the energy consumption in the first place, i. e., which nodes will become active and which nodes will be sleeping. We used a simplified energy model that only considers the energy consumed by turning the nodes' radio transceivers on and off. The energy consumed by transmitting and receiving was neglected because the additional costs are marginal compared to the cost for idle listening. Furthermore, note that the simulation time is considerably shorter than the lifetime of real sensor nodes. Thus, concerning the actual node lifetime, the number of packet transmissions will be low because nodes will sleep most of the time. Other MAC layer issues for minimizing idle listening are not considered as they are complementary to the topology management and can be used in addition.

An overview of the simulation parameters used is given in Tables 7.1 and 7.2. TECA was run with $\alpha = 0$, $\beta = 0$, and $PV = 0.8$, representing an appropriate trade-off between optimizing the packet delivery ratio and optimizing the link lifetime. Once per second, active and passive nodes sent an announcement packet that contained the node's id, state, energy, timeout, and link loss information about its active neighbors. To keep neighbor information consistent, a sequence number was included, too. Nodes are considered dead if an announcement has not been received within $T_d = 10$ s. To reduce the computational complexity, nodes did not verify their states each time a control packet was received but set a timer first. The appropriate verification timeout T_v was set to 200 ms. When it expired, the node's state was verified, bundling all neighborhood changes that occurred within T_v. The passive timeout T_p after a node switches from passive to sleep was two seconds. How long a node sleeps (T_s) depends on the cluster timeout factor and on the timeouts of adjacent nodes. It is calculated by each node individually according to Equation 7.8. To get first link loss estimates, an initialization phase of ten seconds (T_i) is carried out before the actual simulation is started.

Simulation parameter	Setting
Simulation area	100×100 m^2
Number of simulation runs	200
80% radio transmission range	24.3 m
Loss threshold LT	0.8
Number of retransmissions	3
T_{init} (T_i)	10 s
$T_{announcement}$ (T_a)	1 s
$T_{passive}$ (T_p)	2 s
T_{verify} (T_v)	200 ms
T_{dead} (T_d)	10 s

Table 7.1: Simulation settings

Simulation parameter	Setting
Simulation time [s]	$1 \ldots 10^5$
Node density μ	$10 \ldots 50$
Initial energy T^{energy} [s]	$10^2 \ldots 10^5$
Cluster timeout factor γ	$0.1 \ldots 1.0$

Table 7.2: Varied simulation parameters

With r being the radio transmission range of a node, GAF uses a grid cell size of $r/\sqrt{5}$. According to the packet reception model discussed in Section 4.3.1 of Chapter 4, the cell size must be about 10 m in order to achieve a connectivity of about 80% between adjacent cells. Like TECA, GAF employs the same timeouts: T_i, T_a, T_p, T_v, and T_d. However, the sleeping timeout T_s is calculated differently: As each grid cell can be considered to be a cluster itself, with the active node being the cluster head that computes the cluster timeout T_c like in TECA, T_s is equal to T_c. Furthermore, a node goes to sleep as soon as it has encountered another node with more energy. Announcement packets do not include 1-hop neighborhood information because packet reception ratios are not captured. Instead, the nodes' geographical position needs to be included.

Like TECA, ASCENT relies on link measurements concerning the reception quality, and thus uses a similar packet structure. The sleeping timeouts are calculated by considering the timeouts of all active neighbors, which set their own timeouts according to Equation 7.1 (like cluster heads in TECA). Thus, T_s is the minimum of all active neighbor's timeouts. The neighbor threshold NT controlling ASCENT's topology is set to five, as recommended by the authors in [50]. Other parameters were used in the same way as in TECA.

In RAND, all nodes use a random variable to decide whether or not they operate in an active or in a sleeping mode. Only if a node cannot find an active neighbor will it stay active itself. Hence, each node (whether active or sleeping) should have a neighbor that is part of the backbone topology if the probability of being active is high enough. For the simulations, we set the activation probability to 0.25, which is almost the same level that TECA achieves at a density of 20 nodes. Thus, in contrast to ASCENT, the number of active nodes increases if more nodes are deployed within the network.

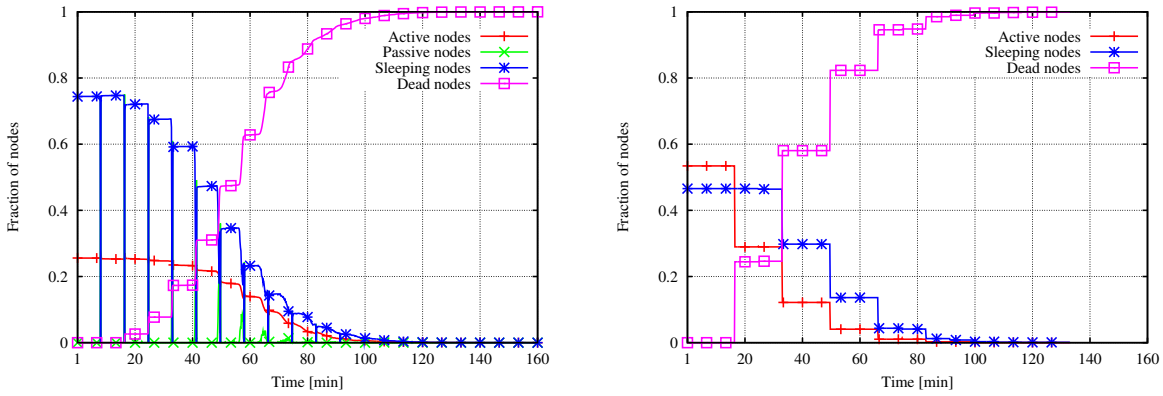
7.5.2 Network Performance over Time

In the first simulation scenario, we considered the selection of active nodes by TECA, GAF, ASCENT, and RAND and their evolution over time, setting the node density μ to 20. We used an initial node energy value sufficient for 1,000 s, i. e., after 1,000 seconds a node will die if it keeps its radio turned on all the time. This represents an initial energy of 960 J according to the energy consumption of an ESB node as defined in Section 4.3.3 of Chapter 4. The cluster timeout factor γ was set to 0.5, balancing network lifetime and energy consumption evenly among all nodes. Due to an initial energy value of

1,000 seconds, sleeping nodes then wake up approximately every 500 seconds in order to balance the energy consumption and to check whether or not the topology had changed in the meantime. Since all nodes were powered on almost at the same time, these cycles are thus more or less synchronized. The following plots therefore contain some peaks, respectively show a "steplike" behavior.

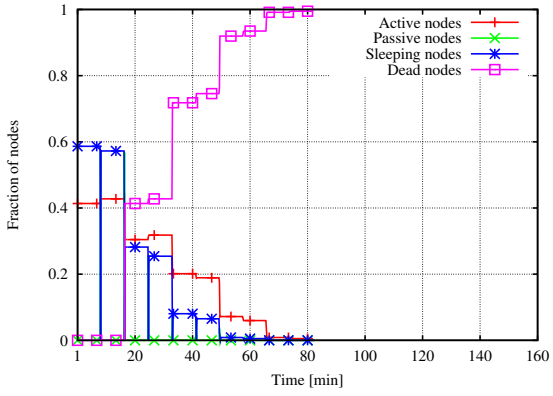
Fraction of Different Node Types

Figure 7.11 depicts the percentage of nodes that were active, passive, sleeping, or dead over time for TECA, GAF, ASCENT, and RAND, respectively. As discussed before, active nodes are nodes the backbone topology consists of, i. e., in TECA, cluster heads and bridges. Passive nodes will still have their radios on and will probe the network, becoming active if necessary. After the passive timeout has expired, they will go into the sleep mode, turn their radios off and change back to passive after the sleeping timeout T_s. If the energy of a node has been consumed completely, the node will "die" and will no longer be available in the network.

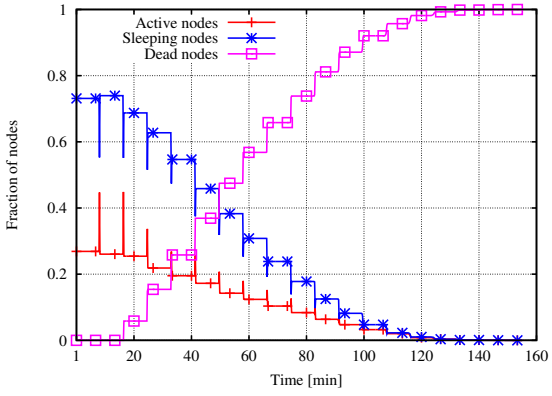


(a) TECA

(b) GAF



(c) ASCENT



(d) RAND

Figure 7.11: Fraction of different node types ($\mu = 20$, $\gamma = 0.5$, $T^{energy} = 1,000$ s)

At the beginning of the simulation, TECA, as well as RAND, selected fewer active nodes, which led to more nodes that slept and saved energy. GAF and ASCENT required more active nodes because of a small grid cell size and a high neighbor threshold, respectively. If the states of cluster heads in TECA changed, it took some time until the topology was stable again. In this case, some nodes

needed to stay passive, to decide whether or not to become active. The time span until the topology was stabilized depended on the announcement time T_a, the time nodes stayed passive, and on how fast nodes verified their state after neighborhood changes. However, even if TECA needed more time to settle, Figure 7.11(a) shows that the number of passive nodes is often very small. The total number of active nodes was almost constant until the majority of nodes had run out of energy. On the average, after 100 minutes, almost all nodes were dead, although the maximum lifetime over all simulation runs lasted for about 160 minutes. Thus, it should be noted that all time plots show average values per *time unit* and indicate the maximum time until all nodes were dead.

Since GAF is based solely on the number of selected nodes in a grid cell, the number of active nodes was significantly higher than for TECA. As long as not all nodes within a cell were dead, the number of active nodes was constant. However, as soon as entire cells died, the number of active nodes decreased quickly. Passive nodes were not required since once a node had discovered a neighbor with more energy within the same cell it went to sleep.

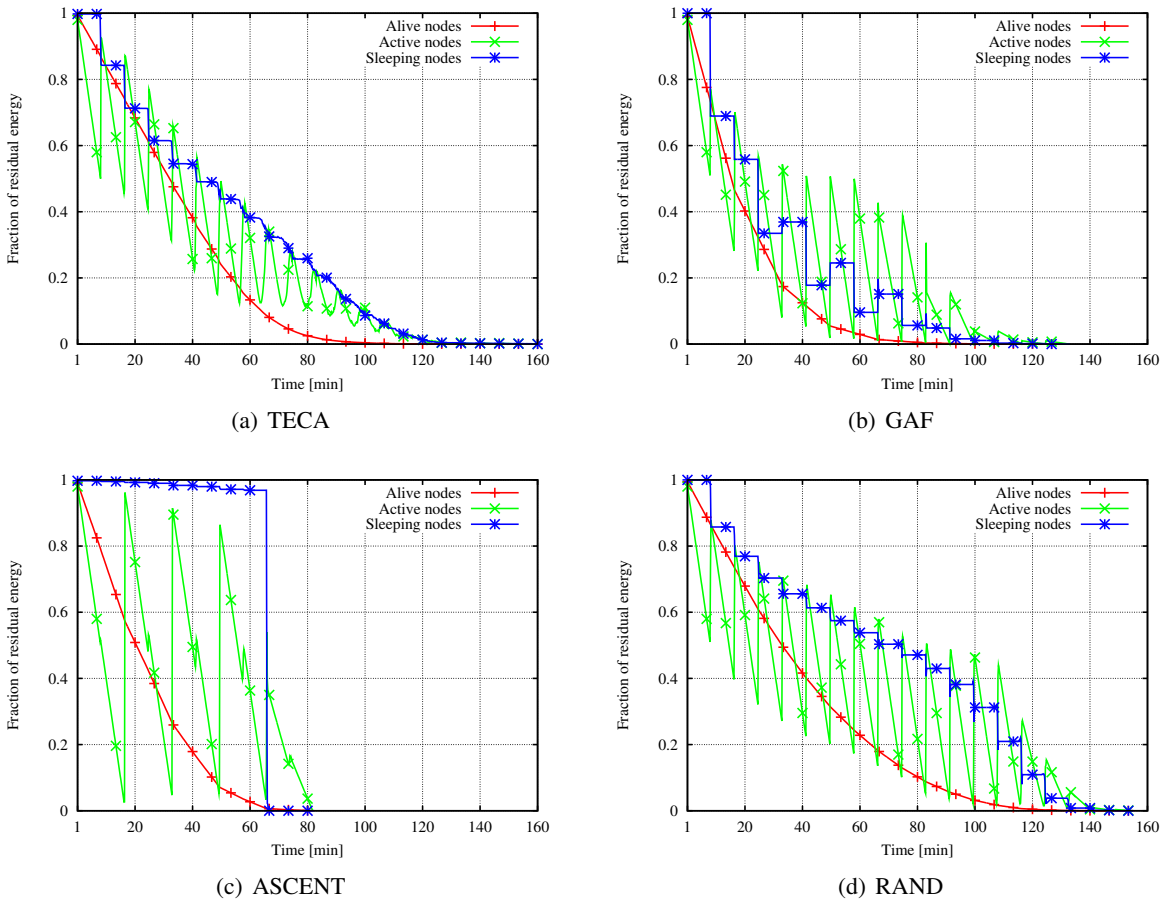
Although ASCENT kept the number of active neighbors of non-dead nodes constant, the total number of active nodes decreased as soon as nodes ran out of energy. In the same way, the number of active nodes in RAND dropped, as the expectation value of active nodes decreases if the number of dead nodes increases. The fraction of dead nodes shows that nodes in TECA and RAND stayed alive longer than in GAF and ASCENT. However, closer to the end, TECA selected more nodes than RAND in order to prevent network partitions, leading to a shorter lifetime until all nodes were dead. Hence, concerning the average network lifetime, RAND outperformed all other topology algorithms.

Fraction of Residual Energy

Figure 7.12 depicts the residual energy averaged over all alive nodes, active nodes, and sleeping nodes. While sleeping nodes do not consume energy, the energy of active nodes constantly decreases until the topology is rebuilt. After rebuilding the topology, nodes with little remaining energy went to sleep, and nodes sleeping before became active. Thus, the residual energy averaged over active nodes shows several peaks. Since in ASCENT nodes stayed active until they ran out of energy, the energy balance among all nodes was worse, which is indicated by a large deviation between the residual energy of sleeping and active nodes. A comparison of all algorithms shows that RAND ran for about 64 minutes, TECA for 54 minutes, ASCENT for 39 minutes, and GAF for only 30 minutes until the average residual energy fraction was 0.2. Although RAND achieved a longer lifetime, TECA balanced energy among nodes much better. This is due to its cluster selecting process, which takes the residual energy values of nodes into account, rather than selecting nodes randomly like in RAND.

Network Partitions

Concerning the "quality" of the topology, RAND may suffer from bad packet delivery ratios due to network partitions. Even if the number of active nodes in TECA and RAND was almost the same at the beginning of the simulation (see Figures 7.11(a) and 7.11(d)), arbitrarily selecting active nodes may be insufficient if the probability of becoming active is too small. This issue is shown in Figure 7.13, which



(a) TECA (b) GAF

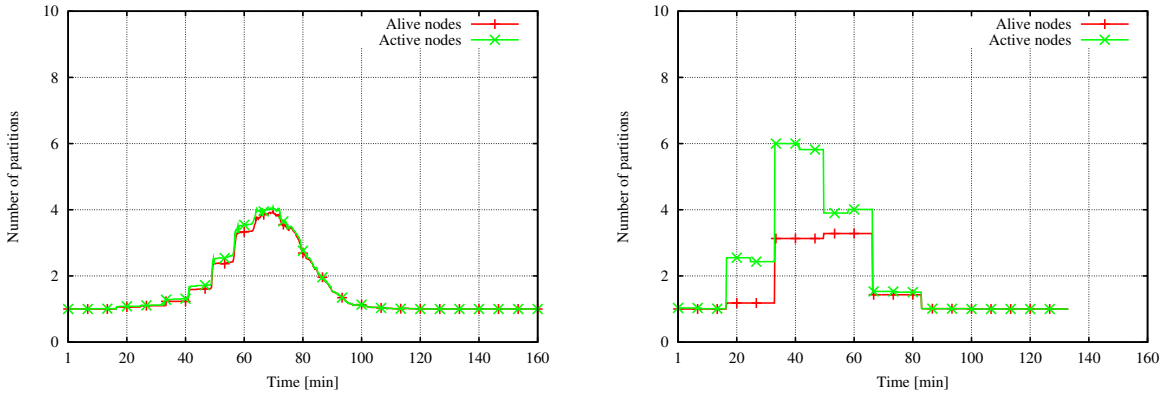
(c) ASCENT (d) RAND

Figure 7.12: Fraction of residual energy ($\mu = 20$, $\gamma = 0.5$, $T^{energy} = 1,000$ s)

depicts the average number of network partitions over time. In order to provide a fair comparison of all algorithms, not only the number of partitions regarding the overlay backbone topology but also regarding the raw topology (that consists of all nodes alive) is shown. By taking all alive nodes into account, the number of partitions provides a lower bound for each algorithm considered. Thus, guaranteeing connectivity requires that the number of partitions be the same in both cases.

As Figure 7.13(a) illustrates, TECA maintained the network connectivity very well and outperformed GAF, ASCENT, and RAND clearly. The connectivity of TECA's topology was almost optimal over the entire simulation time. The first increase and later decrease of the number of partitions can be explained as follows: As long as many nodes were alive, nodes which ran out of energy caused disconnections after some time and thus increased the number of partitions. However, once all nodes of a partition were dead, a partition "vanished" again. Thus, the number of partitions decreases at the end of the network's lifetime.

Since neither GAF, ASCENT, nor RAND took the network connectivity into account explicitly, it is not surprising that the network is partitioned from time to time, although the raw topology was not partitioned. However, considering the small-size simulation area, the number of partitions was substantial and would likely be higher if the size of the network increased. Although a smaller grid size used by GAF, a higher neighbor threshold used by ASCENT, or a larger activation probability used



(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.13: Number of network partitions ($\mu = 20$, $\gamma = 0.5$, $T^{energy} = 1,000$ s)

by RAND might have led to more resistant topologies with regard to network partitions, note that then also less energy could be conserved. As a consequence, the operational lifetime of the network would be shorter.

Independent of such parameters, TECA maintained the network connectivity much better. However, *guaranteeing* connectivity requires full knowledge of any node about its 2-hop neighborhood. Thus, short-term partitions may sometimes occur, because

- it may take some iterations until the topology is stable,
- nodes may go to sleep before they know about new clusters around,
- inconsistency in neighbor tables may exist due to lost announcement packets, or
- estimates about packet reception ratios may be wrong.

The first two issues could be solved by larger passive state timeouts and shorter announcement intervals, reducing the time until TECA has established a stable topology. However, in the majority of cases, partitions will likely occur due to neighbor table inconsistencies if information about state changes of adjacent nodes gets lost. In such a case, nodes might make wrong decisions about whether

or not to become active. In order to minimize such situations, as well as to reduce the probability of lost packets, announcement packets were retransmitted three times[5].

Besides, it should be noted that the number of partitions was based on an 80%-connectivity, i. e., nodes were considered connected if their packet reception ratios were greater than $1-LT$ (see Section 7.3.2). Thus, short-term link measurements may lead to wrong estimates concerning packet loss by assuming that two nodes are connected, although their link losses might be larger than LT.

Packet Delivery Ratio

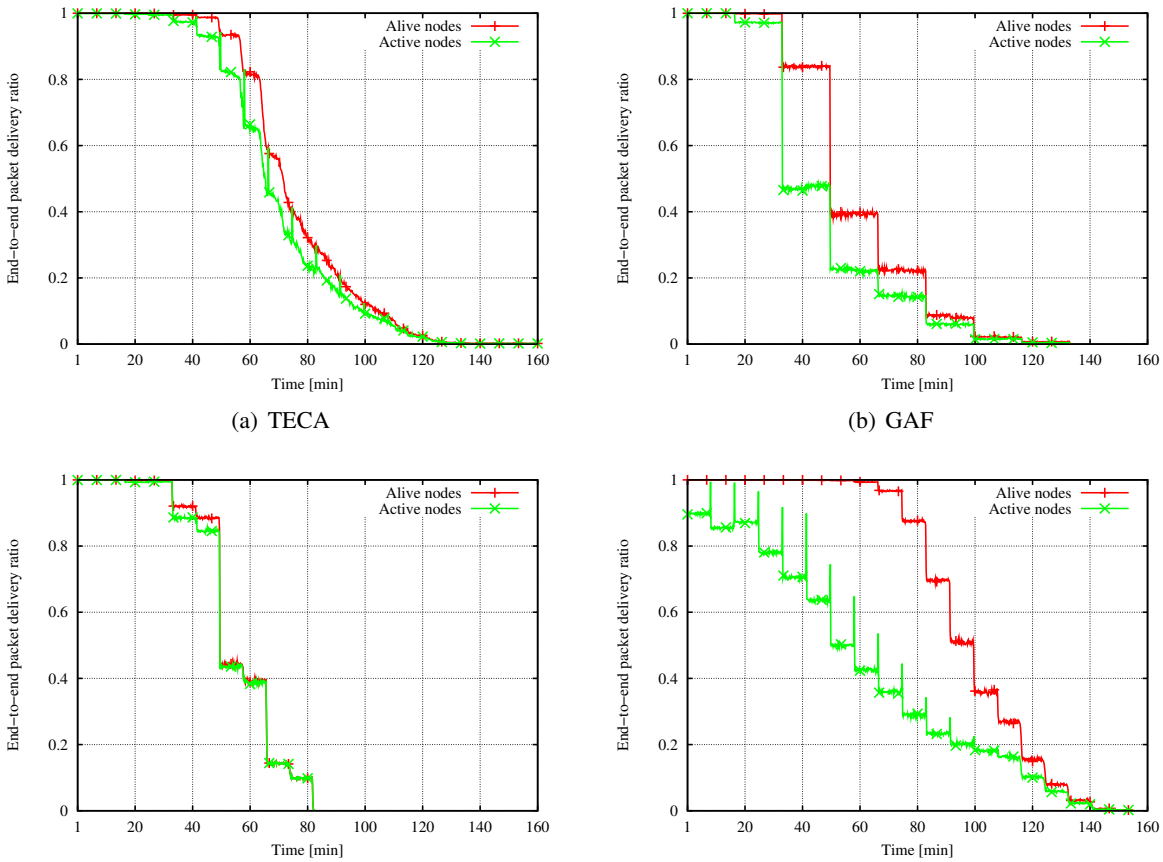
It is also interesting to investigate the overall network quality in terms of packet delivery performance. As no assumption is made about communication patterns or the arrangement of source and sink nodes within the network, we used the following setup: Among all nodes, we randomly selected 100 pairs of nodes, each consisting of an *arbitrary* sending and an *active* receiving node. Packets were forwarded, using up to three retransmissions, by each active node that received a packet correctly. As sleeping nodes had their communications radios turned off, they did not participate in forwarding. If the sending node was sleeping, it woke up temporarily. Network congestion was neglected. Note that if congestion had been taken into account, the performance of GAF and ASCENT might have been worse due to their denser backbone topologies.

By means of this setup, we evaluated the end-to-end packet delivery ratio once per minute. As we focused only on the delivery ratio, delay issues were of no concern. The propagation and the processing time of forwarding packets were thus neglected. Figure 7.14 shows the resulting end-to-end packet delivery ratio averaged over all sender-receiver pairs. As for Figure 7.13, considering all nodes to be alive provides an upper bound for the forwarding process.

Figure 7.14(a) and 7.14(c) illustrate that TECA and ASCENT achieved the highest delivery ratios, which are quite tight regarding their upper bounds. Although ASCENT did not guarantee connectivity, its performance was almost optimal. On the other hand, TECA's delivery ratio was often worse than its upper bound. However, note that the lifetime of TECA was much longer; and note that ASCENT benefited from a dense topology. The topology of TECA was considerably lighter, which minimized the probability of successfully delivering packets to the destination node. Thus, it is assumed that in case of congestion, TECA may perform even better than ASCENT.

Nevertheless, both algorithms took advantage of actively measuring packet reception ratios. While ASCENT considered nodes as neighbors only if their estimated loss rate was below the LT threshold, TECA selected only well-connected bridge nodes to connect cluster heads with each other. However, in order to achieve high end-to-end delivery ratios, not only the quality of established links, but also the number of network partitions is important. While TECA benefited from avoiding network partitions, GAF and RAND suffered from partitions, as well as from bad link connections, because neither algorithm took reception ratios into account. Figure 7.14(b) shows that the assumption made by GAF that every node in a grid cell would be able to communicate with nodes in adjacent cells did not always hold true, partly affecting the end-to-end packet delivery ratio between nodes. On the other hand,

[5]Note that using acknowledgements is not possible as announcement packets are sent by broadcast.



(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.14: End-to-end packet delivery ratio ($\mu = 20$, $\gamma = 0.5$, $T^{energy} = 1,000$ s)

RAND suffered from both a random selection of active nodes and a heavily partitioned network, as shown in Figure 7.14(d). However, in contrast to GAF, the number of active nodes was substantially lower, which though it increased its lifetime, deteriorated its delivery performance considerably.

A comparison of the end-to-end packet delivery ratio among all algorithms is provided in Figure 7.15, which shows a superior performance for TECA. Although TECA's delivery ratio was lower than its upper bound and not as tight as for ASCENT, it outperformed GAF, ASCENT, and RAND most of the time. Not until about 80 minutes had passed did TECA's delivery ratio drop below that of RAND because from this time on nodes ran out of energy quickly.

Again, the packet delivery ratio achieved by GAF, ASCENT, and RAND could be improved, which, however, would have increased their energy consumption and thus would have worsened their network lifetimes. Moreover, finding the right thresholds and parameters will heavily depend on the environment the nodes are placed into, which might change over time. In contrast, TECA did not rely on such conditions and was able to establish a backbone topology that was nearly optimal concerning both the number of network partitions and the packet delivery ratio. Thus, independent of environmental conditions, TECA is expected to achieve good performance results, while connectivity will be (almost always) maintained.

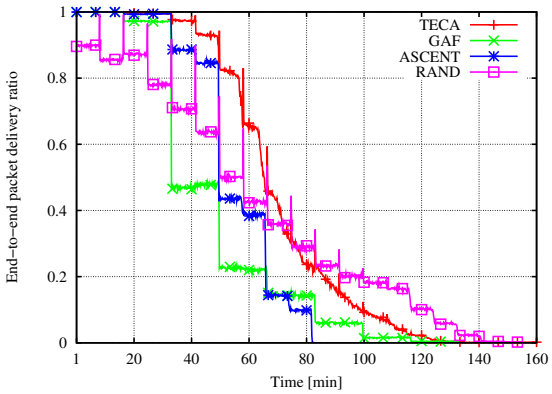


Figure 7.15: Comparison of end-to-end packet delivery ratios ($\mu = 20$, $\gamma = 0.5$, $T^{energy} = 1,000$ s)

7.5.3 Influence of Node Density

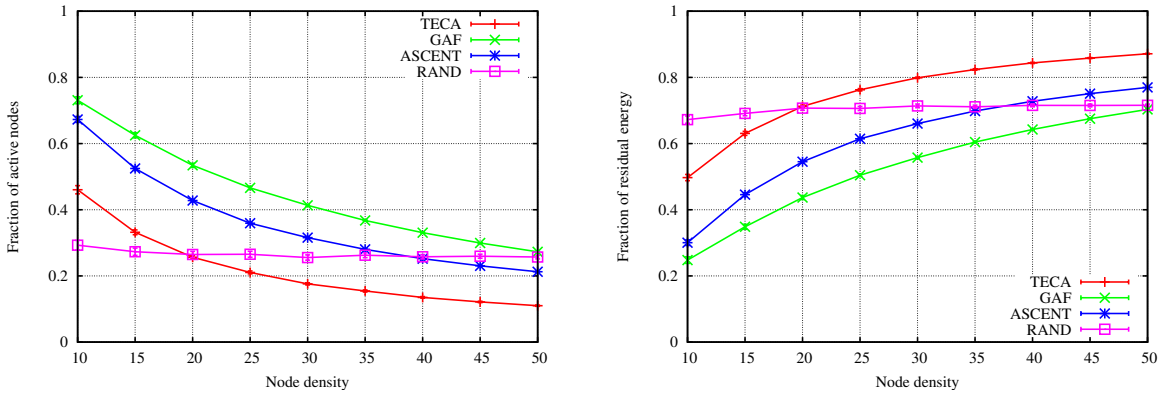
By means of further simulations, we also investigated the influence of different node densities on the performance of TECA, GAF, ASCENT, and RAND. Based on the same simulation settings as before, the following section analyzes how many nodes were be active and how the residual energy changed if the node density increased.

Fraction of Active Nodes and of Residual Energy

Figure 7.16(a) depicts the fraction of active nodes after 100 seconds, averaged over all simulation runs. Until this time, actually each algorithm should have achieved a stable topology. In addition to average values, 0.95 quantiles are shown, which, however, were often too tight to be seen clearly. While all algorithms benefited from higher node densities with respect to the number of active nodes, TECA outperformed ASCENT, GAF, and RAND when the density was larger than 20 nodes. Compared to ASCENT and GAF only, TECA always performed better. The fraction of active nodes used by RAND was nearly 25% due to its activation probability. For low and moderate node densities, it outperformed ASCENT and GAF clearly, which rather tried to keep the *number* of active neighbors constant. GAF required many active nodes due to its constraint that all nodes in adjacent grid cells should be able to communicate with each other; thus, the size of a grid cell needed to be quite small. On the other hand, ASCENT performed better because nodes slept when they found enough active neighbors.

Hence, if the density of nodes was high enough, TECA achieved the best result, as it was more adaptive than ASCENT, GAF, and RAND. Because redundant nodes went to sleep as soon as they discovered that the backbone topology was well connected, fewer nodes were active. Thus, reliance on a *fixed* fraction or on a fixed number of active nodes could be avoided.

Due to the differences in the number of active nodes, there were significant differences in the overall energy consumption of the network, as shown in Figure 7.16(b). This figure depicts the fraction of residual energy averaged over all nodes after 1,000 seconds, i. e., after the lifetime of a single node that never slept. Because TECA performed best concerning the number of active nodes for node densities



(a) Fraction of active nodes ($t = 100$ s)

(b) Fraction of residual energy ($t = 1{,}000$ s)

Figure 7.16: Fraction of active nodes and of residual energy ($\gamma = 0.5$, $T^{energy} = 1,000$ s)

larger than 20, it saved the most energy. Thus, TECA can be expected to prolong the network's operational lifetime most if the node density increases.

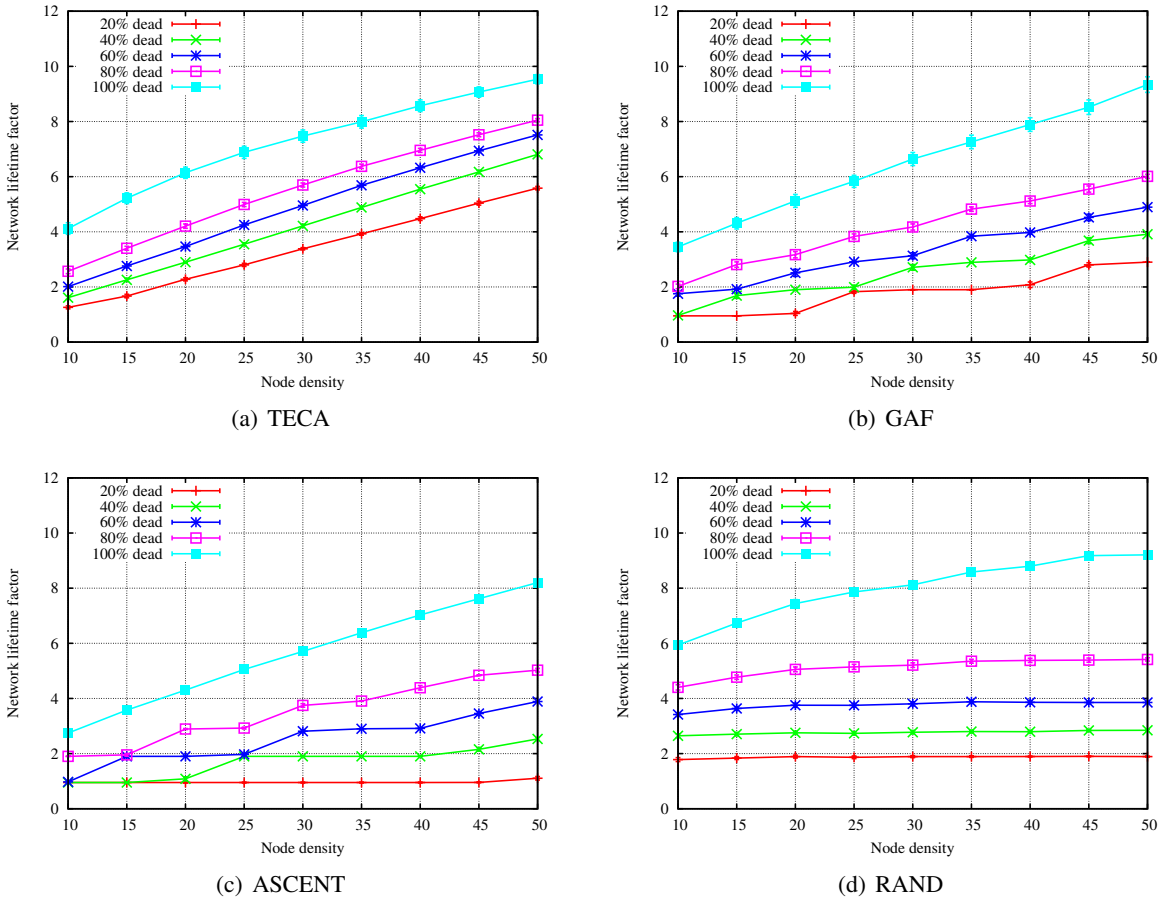
Network Lifetime Factor

We define the network lifetime as the time that expires until 80% of all nodes have died, rather than as the time until the first node runs out of energy, as considered in Chapter 5. As the network may still be useable even if some nodes have dropped out, this should be an adequate percentage in order to reflect the network useability. In particular, note that the aim of the topology management is to exploit a high *redundancy* of nodes by maintaining a network consisting of only some nodes. Thus, nodes are intended to die during the network lifetime anyway.

In order to be independent of the amount of initial energy, we analyzed the *network lifetime factor*, which is defined as the ratio of *network lifetime* and *initial energy* in time units. Figure 7.17 shows the network lifetime factor for different fractions of dead nodes and node densities, using an initial energy of 1,000 seconds, as before. Due to higher energy savings, each algorithm benefited from a greater node density and improved the network lifetime factor accordingly.

Concerning the performance of all algorithms, TECA was able to achieve the best energy balance among all nodes. This is indicated by the even increase in the lifetime factor, independent of the fraction of dead nodes. Thus, the majority of nodes stayed alive very long, as shown in Figure 7.17(a). Since GAF required one active node per grid cell, cells with only few nodes died early. Hence, its network lifetime factor for a fraction of 20% of dead nodes was much smaller (see Figure 7.17(b)). As shown in Figure 7.17(c), ASCENT's energy balance was also worse because all nodes stayed active until they had run out of energy. The energy balance of RAND was not much better although nodes became active randomly. However, since residual energy was not taken into account, RAND was also outperformed by TECA regarding the time the first nodes took to consume their available energy.

Figure 7.18 provides a better plot to compare TECA, GAF, ASCENT, and RAND, showing the network lifetime factors for the case of 80% of dead nodes. We see that as long as the density was less



(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.17: Network lifetime factor ($\gamma = 0.5$, $T^{energy} = 1,000$ s)

than 25 nodes, RAND achieved the longest network lifetime. For higher node densities, it was outperformed by TECA. The network lifetimes of GAF and ASCENT were significantly worse. As long as the density was low, RAND performed better because it kept the fraction of active nodes constant, even when the density decreased to 10 nodes. In contrast, TECA required a larger fraction of active nodes in order to maintain connectivity. Thus, when more nodes were deployed, connectivity was achieved by using a smaller node fraction, which increased the network lifetime of TECA accordingly.

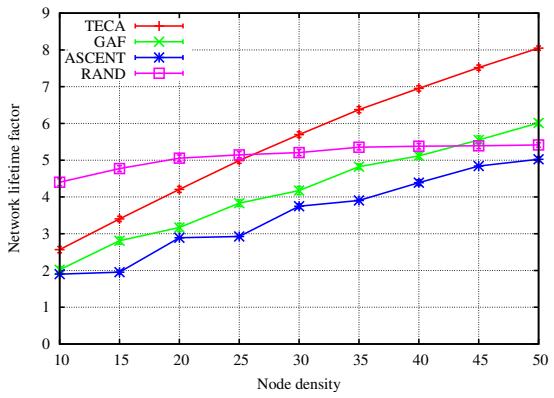
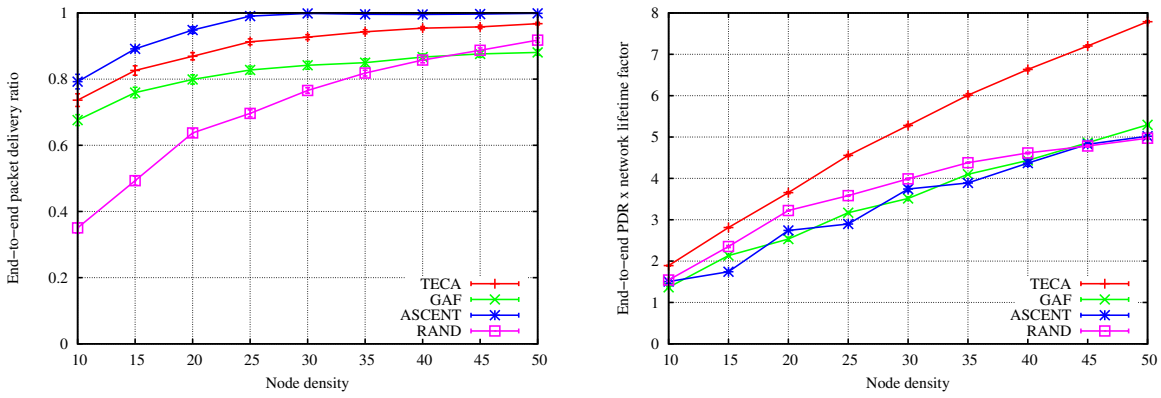


Figure 7.18: Comparison of network lifetime factors ($\gamma = 0.5$, $T^{energy} = 1,000$ s, 80% dead)

Packet Delivery Ratio

As discussed earlier, RAND's higher lifetime factor may have come at the expense of network partitions and worse packet delivery ratios. This is also illustrated in Figure 7.19(a), which shows the end-to-end packet delivery ratio for different node densities, averaged over time until 80% of nodes have died. ASCENT achieved the highest delivery ratio because it was able to maintain a well-connected network for most of its lifetime. But it should be noted that the lifetime of ASCENT was significantly shorter, which may have influenced the *absolute* number of packets delivered considerably. Regarding the performance of TECA, the average delivery ratio was always better than the ratios of GAF and RAND, although TECA's lifetime lasted much longer.

Thus, it may be insufficient to consider solely the packet delivery ratio since each algorithm led to a different network lifetime. To obtain a measure that quantifies the total amount of data delivered, Figure 7.19(b) depicts the product of the delivery ratio and the network lifetime. Combining both metrics shows that the lower delivery ratio of RAND was compensated by a larger network lifetime. The same applies for GAF, which performed similar to ASCENT. However, the performance of TECA was up to about 50% better and thus superior to the performance of GAF, ASCENT, and RAND. This again shows that TECA trades off the different performance metrics very well.



(a) PDR

(b) PDR × network lifetime factor

Figure 7.19: End-to-end packet delivery ratio ($\gamma = 0.5$, $T^{energy} = 1,000$ s, 80% dead)

Finally, the following two sections analyze the influence of the amount of initial energy and the influence of different wake-up times, considering a fixed density of 20 nodes.

7.5.4 Influence of the Initial Energy

The amount of energy a node has at the beginning of the simulation certainly determines the overall network lifetime. Thus, it is expected that the greater the initial energy, the longer the network lifetime will be. However, because the time required to reach a stable topology is independent of the initial energy, the *relative* amount of energy consumed before nodes sleep will increase if the initial energy decreases. This may have a significant impact on the network lifetime, particularly on TECA.

Figure 7.20 depicts the network lifetime factor for initial energy values between 10^2 s and 10^5 s. It is clearly shown that in contrast to GAF, ASCENT, and RAND, the lifetime of TECA changed substantially when the initial energy decreased. This was mainly due to the fact that TECA needed more time to reach a stable topology. As redundant nodes first stayed passive before they went to sleep, the relative energy spent during this time is significant. Thus, the more energy a node had, the less important was the time it stayed passive.

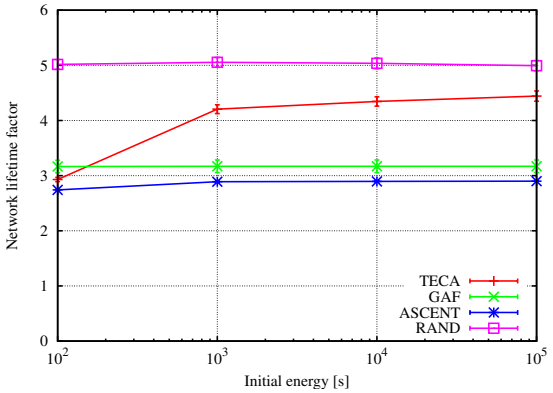


Figure 7.20: Network lifetime factor ($\mu = 20$, $\gamma = 0.5$, 80% dead)

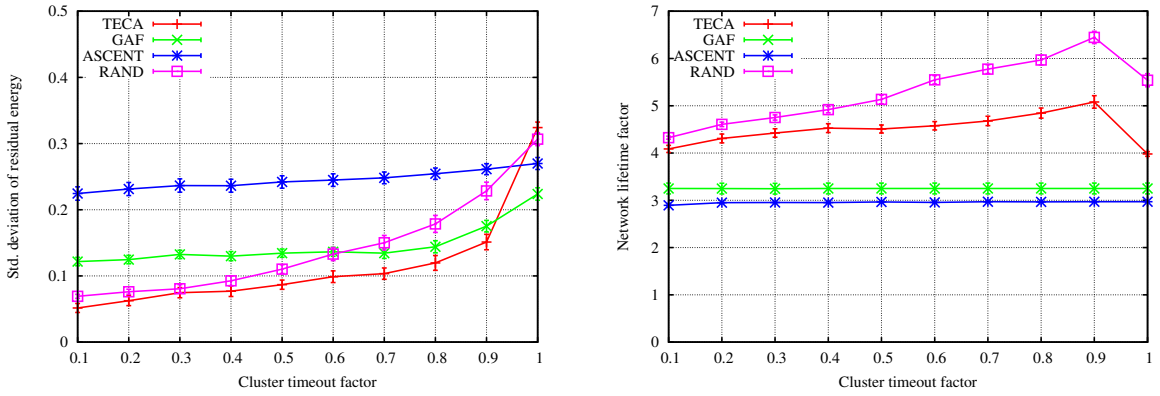
Hence, the impact of the initial energy on GAF, ASCENT, and RAND was rather marginal. The network lifetime increased only slightly if more energy was initially available. Because GAF and RAND terminated quite fast due to their simplicity, they were nearly independent of the initial amount of energy. As ASCENT used passive nodes similar to TECA, its lifetime increased slightly more. However, compared to TECA, the time required by ASCENT for the number of active neighbors to converge towards the predefined threshold was significantly shorter. Nevertheless, although TECA consumed more energy in order to reach a stable topology, its network lifetime was much better than that for either GAF or ASCENT. Only RAND achieved a longer lifetime, but at the expense of a worse network performance, as shown above.

7.5.5 Influence of the Wake-Up Time

In the last set of simulations, we investigated the influence of different timeouts after sleeping nodes had woken up. Note that in GAF, ASCENT, and RAND, each active node was considered a cluster head, such that all sleeping nodes calculated their sleeping timeouts according to Equation 7.1. Thus, we concentrated on the impact of the cluster timeout factor, which was described in Section 7.3.2.

Figure 7.21(a) depicts the standard deviation of the residual energy, calculated at the end of the network lifetime, in order to show how the energy balance among nodes is influenced by different cluster timeout factors. The smaller the standard deviation, the more even is the energy consumption of nodes.

For all algorithms, the energy balance improved if nodes woke up more frequently, i. e., if the cluster timeout factor became smaller. The impact on ASCENT was less significant because active nodes stayed active until they had run out of energy. Only in TECA, GAF, and RAND did active nodes go to sleep as soon as they found adjacent neighbors with more energy. They thus rotated the role of being



(a) Standard deviation of residual energy

(b) Network lifetime factor

Figure 7.21: Influence of cluster timeout factors ($\mu = 20$, $T^{energy} = 1,000$ s, 80% dead)

active more evenly among all nodes, improving the load balance in the network. The fact that GAF performed worse was due to the grid structure used, which balanced the energy consumption per grid cell only. Even if just two nodes in a cell were left, with both nodes having less energy than nodes in adjacent cells, one node was active. Thus, the more evenly nodes were deployed within an area, the better the energy balance of GAF was. As shown in Figure 7.21(a), TECA achieved the best energy balance, since during the construction of the backbone topology, link lifetimes were also taken into account[6].

While the energy consumption in the network will be balanced more evenly if sleeping nodes wake up frequently, the network lifetime will decrease as more energy is spent. This is shown in Figure 7.21(b) which illustrates the influence of different cluster timeout factors on the network lifetime factor. TECA and RAND benefited from higher timeout factors more than did GAF and ASCENT because GAF was restricted to its grid structure and ASCENT suffered from nodes staying active all the time. However, if the cluster timeout factor was one, nodes in TECA and RAND also stayed active until they had run out of energy, resulting in dead nodes sooner. As a consequence, the network lifetime factor dropped, compared to cluster timeout factors smaller than one.

In conclusion, the results of the simulations show that TECA often outperforms GAF, ASCENT, and RAND regarding many performance metrics, in particular regarding the fraction of active nodes, network connectivity, packet delivery ratio, network lifetime, and load balance. The approach taken by TECA thus seems to be reasonable from a theoretical point of view. Whether similar results can be achieved also in a practical implementation is part of the next section.

[6]Note that TECA's priority function as defined in Equation 7.5 is applied for $\alpha = 0$ as well as $\beta = 0$.

7.6 Experimental Evaluation

By running real-world experiments in our WSN testbed, we tried to confirm the results we obtained by means of simulations. We implemented and tested all algorithms on the ESB platform, based on the implementation used in the simulation and evaluated their performances during several experiments.

7.6.1 Experimental Setup

The setting of the evaluation setup is presented in Table 7.3. As described in Chapter 3, our WSN testbed consists of 24 ESB nodes placed in a 4×6 grid structure with a distance between two nodes of about 60 cm. The node's transmission power was set to 15%. The initial energy was set such that each node could stay active for one hour, using the same energy model as before, i. e., nodes would consume energy only if their communications radios were turned on. The cluster timeout factor was set to 0.5 in order to trade off energy balance and topology changes. In contrast to the settings used in the simulation, we had to consider the limited communication and processing capabilities of the ESB nodes. Thus, the initial energy and the following timeouts were increased appropriately. Every ten seconds (T_a), active nodes sent announcement packets, using three retransmissions to account for packet loss. The loss threshold used by TECA and ASCENT was again set to 0.8. Neighbors were considered *dead* if no announcement packets had been received for 50 seconds (T_d), which is equal to losing five announcements successively. Upon receiving announcement packets from neighbors, nodes waited one second (T_v) before they verified their states in order to minimize processing. While the timeouts of active and sleeping nodes were based on residual energies and determined by the cluster timeout factor, the timeout of passive nodes (T_p) was set to 30 seconds (used by TECA and ASCENT only).

Evaluation parameter	Setting
Evaluation runs	10
Number of ESB nodes	24
Initial energy T^{energy}	3,600 s
Cluster timeout factor	0.5
Number of retransmissions	3
Loss threshold LT	0.8
T_{init} (T_i)	60 s
$T_{announcement}$ (T_a)	10 s
$T_{passive}$ (T_p)	30 s
T_{verify} (T_v)	1 s
T_{dead} (T_d)	50 s

Table 7.3: Evaluation settings

In addition to these settings, TECA was run with $\alpha = 0$, $\beta = 0$, and $PV = 0.8$. GAF used a grid cell size of 1.6 m, which led to six cells, of which each consisted of four nodes. The neighbor threshold used by ASCENT was set to five; the activation probability of RAND was 0.3.

In order to obtain statistical information, an additional ESB node was used, called the *evaluation node*. This node was used only to gather information and was not part of the evaluated network. Once per

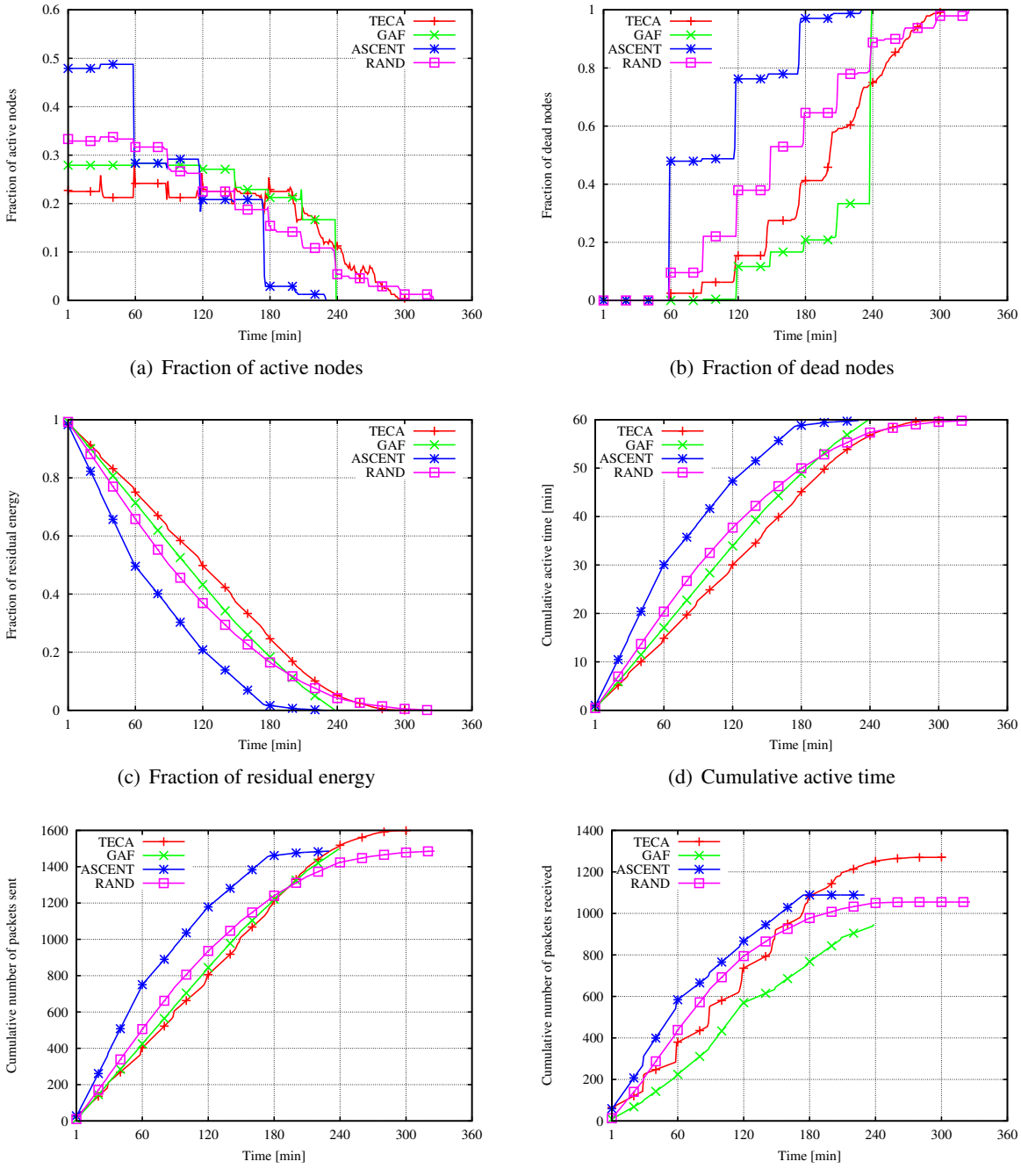
minute, the evaluation node polled every other node in the network and requested state information. For example, whether a node was active or sleeping, for information about its residual energy, or the number of packets sent and received[7]. However, as evaluating the network connectivity over time would have been very time-consuming if each node had sent packets to every other node during the runtime of the experiment, we used the following heuristic: Before the actual evaluation was started, the connectivity between any pair of nodes was measured by means of ping packets. Thus, like in Chapter 4, all nodes broadcast 100 pings, while adjacent nodes captured the number of received packets. Afterwards, the obtained neighbor tables were transmitted to the evaluation node for later processing. A connected computer calculated the appropriate connectivity graphs and evaluated the average connectivity of the backbone topology over time. While network partitions were determined according to the link loss threshold LT, the topology's average end-to-end packet delivery ratio was simulated. As in the previous section, an arbitrary node was selected that sent 100 packets towards an arbitrary active node by enhanced flooding, i. e., already received packets were discarded. Packets were forwarded by active nodes only. To account for packet losses, three retransmissions were employed. The simulations were repeated 100 times to obtain a good estimate of the overall network connectivity.

7.6.2 Evaluation Results

We now present the experimental results, which are averaged over ten evaluation runs. Figure 7.22 gives an overview of several performance metrics measured over time. The fractions of active and dead nodes of all algorithms are shown in Figures 7.22(a) and 7.22(b), respectively. As long as less than 40% of nodes were dead, the topology of TECA consisted of about five to six active nodes. GAF activated about seven nodes and thus one node more than the number of grid cells. Hence, the assumption that all nodes being part of the same cell are connected to each other does not hold true. Also, RAND used slightly more active nodes than specified by its activation probability. This may either be due to the node's random number generator or due to the fact that not all nodes found active neighbors, in which case they became active themselves. With ASCENT, the fraction of active nodes started at about 50%. Thus, almost half the nodes did not find five active neighbors whose packet reception ratios were better than 20%. But note that due to the small network size, around two-thirds of all nodes are border nodes, decreasing the average number of neighbors clearly. As a consequence, ASCENT's network lifetime was shorter than those for TECA, GAF, and RAND. Furthermore, the fraction of dead nodes increased heavily as soon as nodes had run out of energy, as shown in Figure 7.22(b)

The higher energy consumption of ASCENT is also illustrated in Figure 7.22(c), which depicts the fraction of residual energy averaged over all nodes. Due to fewer active nodes, TECA performed best (most of the time) and consumed far less energy. However, in the end, RAND achieved a longer operation time because its expectation value of active nodes decreased if nodes ran out of energy. The best energy balance is achieved by GAF, which rotated the role of being active quite evenly among all nodes and thus showed little variation in its operation time. After 240 minutes, the remaining

[7] It was therefore necessary not to turn off the radio transceiver of sleeping nodes. However, that did not effect the "artificially" calculation of the energy consumption, which solely considered the state of a node.



(a) Fraction of active nodes

(b) Fraction of dead nodes

(c) Fraction of residual energy

(d) Cumulative active time

(e) Number of packets sent

(f) Number of packets received

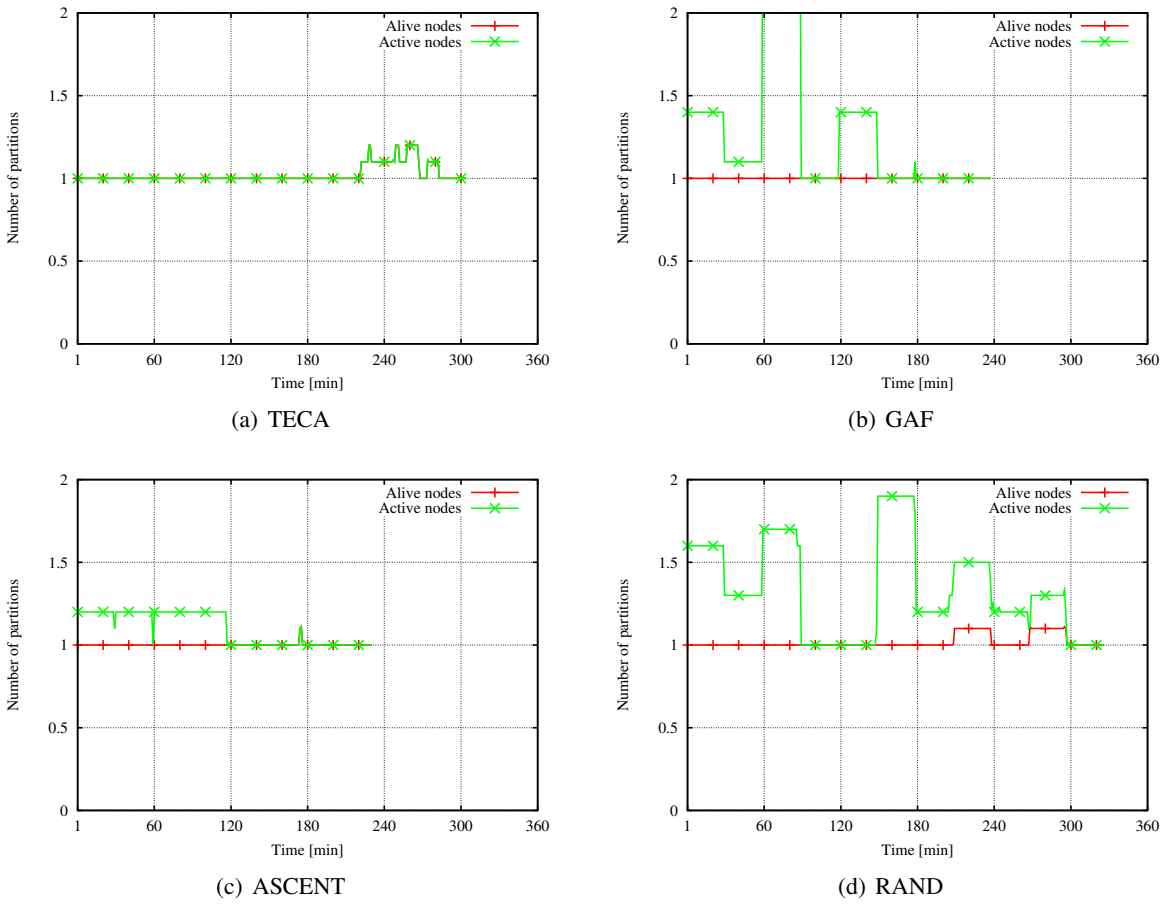
Figure 7.22: Overview of several evaluated performance parameters

nodes ran out of energy almost at the same time (see Figure 7.22(b)). Corresponding to the fraction of residual energy, Figure 7.22(d) shows the cumulative times in which nodes stayed active, respectively in which nodes slept. For example, after one hour, nodes running TECA had slept about 45 minutes, in contrast to only 30 minutes for those running ASCENT.

We also measured the average numbers of announcement packets sent, respectively received correctly, which are illustrated in Figures 7.22(e) and 7.22(f) over time. Concerning the number of sent

announcements, TECA and GAF performed similarly. Thus, although TECA needed more packet exchanges until it reached a stable topology, this overhead was compensated by a lower fraction of active nodes. Moreover, compared to ASCENT and RAND, it even performed better. However, regarding the number of packets received correctly, GAF achieved the best performance. Due to its grid structure used, the GAF algorithm was able to terminate rapidly, putting nodes to sleep very quickly. In addition, it did not use passive nodes like TECA and ASCENT to improve its topology. Since the number of received packets will be the higher the more nodes have their communication radios turned on, TECA was affected most, showing a high increase of received packets when nodes woke up. However, afterwards, it benefited from a better backbone topology, which consisted of fewer active nodes and thus caused a lower energy consumption.

How the network connectivity was influenced by TECA, GAF, ASCENT, and RAND is shown in Figures 7.23 and 7.24, which present the average number of partitions and the end-to-end packet delivery ratio over time. As mentioned above, both metrics were evaluated by simulations, using state information captured during the experiments. As depicted in Figure 7.23(a), TECA's topology was always connected, even at the end of its operation time[8]. In contrast, the topologies of GAF,

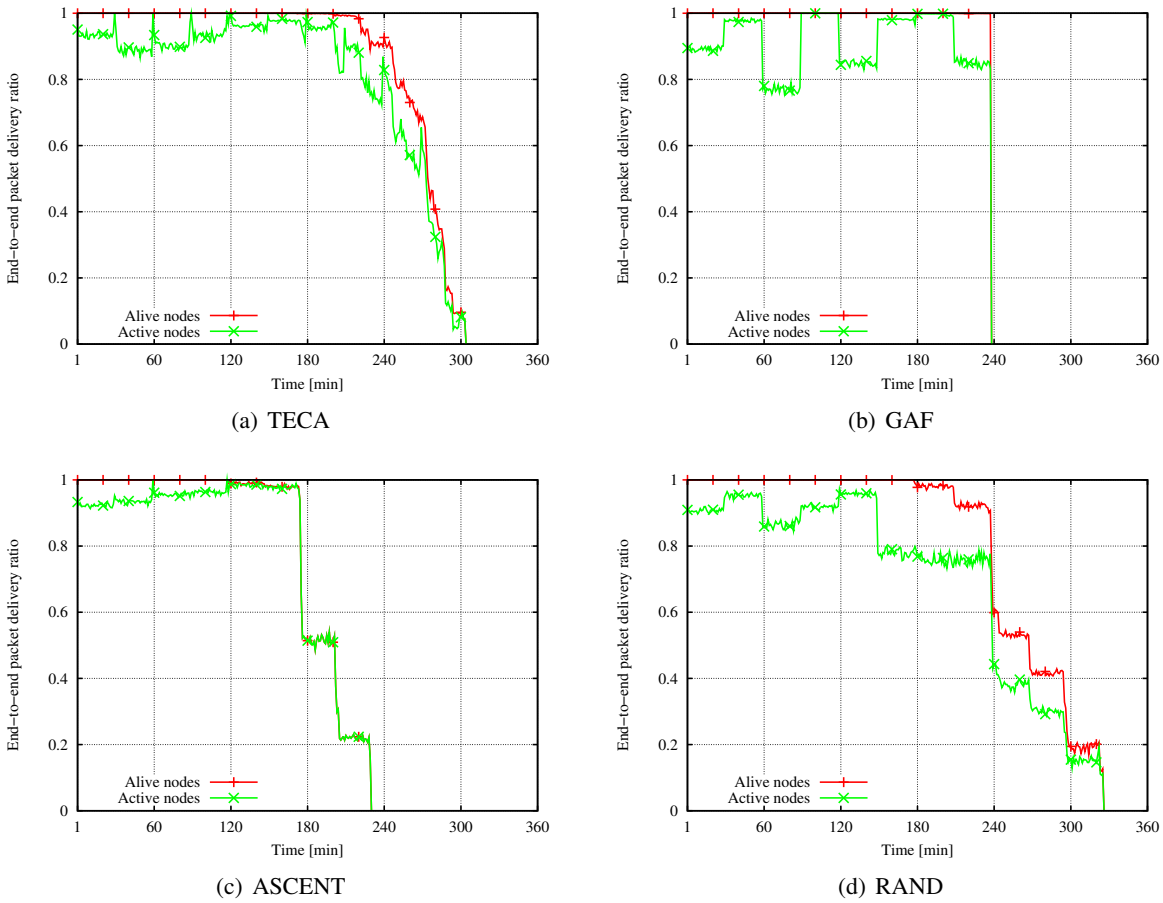


(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.23: Measured number of network partitions

[8]That is, the connectivity graph does not contain links with packet reception ratios smaller than $1 - LT$, respectively 20%.



(a) TECA (b) GAF

(c) ASCENT (d) RAND

Figure 7.24: Evaluated end-to-end packet delivery ratio

ASCENT, and RAND were partitioned occasionally, although more nodes were active. However, the number of partitions was quite low due to the small network size. Compared to GAF and ASCENT, RAND suffered from disconnections more frequently. But it should be noted that GAF benefited from an even distribution of active nodes by exploiting geographic information, while ASCENT benefited from a higher fraction of active nodes.

Due to the relatively small-sized testbed, partitions had only little impact on the network performance, as shown in Figure 7.24. All algorithms achieved acceptable packet delivery ratios; often, the delivery ratios were even nearly 100%. If, in addition, the operation time of each algorithm is taken into account, TECA showed the best performance, since its delivery ratios remained high for a long time. While GAF and ASCENT performed worse due to shorter lifetimes, RAND suffered from worse packet delivery ratios, although its maximum operation time was about 10% longer.

Finally, Table 7.4 shows the lifetime factors and packet delivery ratios for different fractions of dead nodes, averaged over time until the appropriate fraction of nodes was dead. In addition to average values, the corresponding 0.95 t-quantiles are shown, too, identifying the variation among the different evaluation runs. As shown in the fourth row (80% dead nodes), the network lifetime of TECA was about 43% longer than that of ASCENT, and about 4% longer than those of GAF and RAND. If we consider the time that elapsed until all nodes ran out of energy, RAND performed about 7% better than

Performance metric		TECA	GAF	ASCENT	RAND
Network lifetime factor	20% dead	2.29 [2.03, 2.55]	2.73 [2.45, 3.01]	0.98 [0.98, 0.98]	1.73 [1.52, 1.94]
	40% dead	2.89 [2.54, 3.24]	3.97 [3.97, 3.97]	0.98 [0.98, 0.98]	2.37 [2.00, 2.74]
	60% dead	3.60 [3.32, 3.89]	3.97 [3.97, 3.97]	1.96 [1.95, 1.96]	3.08 [2.66, 3.49]
	80% dead	4.14 [3.90, 4.37]	3.97 [3.97, 3.97]	2.90 [2.89, 2.91]	3.97 [3.55, 4.39]
	100% dead	4.57 [4.35, 4.79]	3.97 [3.97, 3.97]	3.43 [3.10, 3.75]	4.90 [4.46, 5.34]
Packet delivery ratio	20% dead	0.98 [0.97, 0.99]	0.96 [0.94, 0.97]	0.96 [0.94, 1.00]	0.96 [0.92, 0.99]
	40% dead	0.98 [0.97, 0.99]	0.95 [0.94, 0.96]	0.96 [0.94, 1.00]	0.95 [0.92, 0.97]
	60% dead	0.97 [0.96, 0.99]	0.95 [0.94, 0.96]	0.97 [0.94, 1.00]	0.93 [0.92, 0.95]
	80% dead	0.95 [0.94, 0.97]	0.95 [0.94, 0.96]	0.98 [0.96, 1.00]	0.89 [0.86, 0.93]
	100% dead	0.84 [0.81, 0.88]	0.95 [0.94, 0.96]	0.84 [0.78, 0.91]	0.77 [0.70, 0.83]
Packet delivery ratio × network lifetime factor	20% dead	2.24 [1.96, 2.53]	2.61 [2.31, 2.91]	0.94 [0.89, 1.00]	1.65 [1.40, 1.92]
	40% dead	2.83 [2.45, 3.21]	3.77 [3.72, 3.82]	0.94 [0.89, 1.00]	2.24 [1.85, 2.66]
	60% dead	3.51 [3.18, 3.84]	3.77 [3.72, 3.82]	1.90 [1.82, 1.97]	2.87 [2.44, 3.32]
	80% dead	3.94 [3.67, 4.22]	3.77 [3.72, 3.82]	2.84 [2.76, 2.91]	3.54 [3.04, 4.08]
	100% dead	3.86 [3.50, 4.23]	3.77 [3.72, 3.82]	2.89 [2.41, 3.42]	3.75 [3.11, 4.45]

Table 7.4: Evaluated network lifetime factors and packet delivery ratios

TECA, achieving a lifetime factor of 4.9. Concerning the topologies' average packet delivery ratios, ASCENT outperformed all other algorithms as long as the dead node fraction was less than 80%. However, the smaller lifetime factor of ASCENT should not be neglected. Even if the packet delivery ratio was higher, the total number of delivered packets may have been much smaller. Combining the lifetime factor and the average packet delivery ratio per time unit is thus quite reasonable, as it indicates over which topology more packets may be transmitted. Then, ASCENT performed even worse due to the shortest network lifetime, as shown in the last five rows of Table 7.4. On the other hand, TECA achieved a superior performance, which verifies our simulation results from Section 7.5.

7.7 Conclusions

In this chapter, we have presented a new topology and energy control algorithm that establishes a backbone topology of active nodes, taking into account the residual energies and packet delivery ratios of nodes. In addition to active nodes, TECA identifies redundant nodes, which are not required to maintain connectivity. While active nodes will keep their communication radios turned on, redundant nodes will switch to a low-power energy mode and turn their radios off completely. In so doing, a significant amount of energy can be conserved, extending the network lifetime substantially.

One of the design issues of TECA has been network connectivity, which is achieved by calculating minimum cluster spanning trees. Nodes that are part of these spanning trees will act either as cluster heads or bridges to connect different clusters with each other. By using only local information, TECA operates in a fully distributed manner. Furthermore, it is able to adapt to different environments and application requirements by accounting for node densities, packet losses, and link lifetimes.

Results from both simulations and real-world experiments have proven the superior performance of TECA, compared to three other approaches proposed in the literature. Although the experiments were carried out on a small-sized network consisting of only a few nodes, the results are very interesting, as they indicate a similar network performance to the one obtained by means of simulations. Despite a

higher overhead, TECA performs best and trades off the number of active nodes, energy consumption, network connectivity, network lifetime, as well as the packet delivery ratio very well.

For future work, it would be advisable to augment TECA also by MAC-related features, like scheduling of packet transmissions, as well as of active and sleep times, in order to reduce idle listening. In so doing, even active nodes will benefit from lower duty cycles and conserve energy most of the time. Furthermore, cluster heads could be used to manage the access to the wireless channel by means of TDMA (time division multiple access). In addition, they could easily be used to aggregate data, as packets are likely to be sent to a cluster head first.

Although TECA is a complementary approach to the forwarding strategies we considered in the preceding chapters, analyzing the combination of topology management and energy efficient forwarding would also be an interesting aspect of future work. In particular, the priority function of TECA could be augmented by information regarding energy-efficient forwarding paths, lifetime efficiency, and data aggregation.

That actually concludes the scientific contributions of this thesis. In the following chapter, we will finally present several application scenarios which are already in use, either by research or industrial institutions. Afterwards, Chapter 9 will summarize the thesis and its major contributions.

CHAPTER 8

Sensor Network Applications

"Computers do not solve problems – computers carry out solutions, specified by people, to problems."

– D. D. Spencer –

8.1 Introduction

In the last few years, there has been a vast number of application proposals and deployments for sensor networks [14, 116]. While some application scenarios have not yet been fully evaluated in practice and so far are based only on prototype systems, other applications have already been used for years. For researchers, particularly the huge variety of possibilities for collaborations between the *sensing*, *processing*, and *actuating* performed by dozens or hundreds of very small, low-powered sensor nodes offers tremendous opportunities, making sensor networks so interesting and challenging.

As most sensor nodes are battery-powered, and recharging batteries is often difficult, almost each application has to deal with issues like minimizing the energy consumption of the sensor network. Achieving a high energy efficiency on each layer of the network stack is thus very important. Therefore, the algorithms proposed in this thesis can be used, as they perfectly fit into the context of many application scenarios. While energy-efficient forwarding and aggregation may be used for the data gathering process, TECA may be used to minimize the energy consumption of idle listening as well as to establish a well-connected forwarding network.

Although research regarding energy efficiency is one of the most interesting aspects of sensor networks, the development of easy to use deployments has recently also become of great interest in terms of applications. E. g, Greenstein *et at.* [101] have proposed a generic *sensor network application construction kit* (SNACK), which consists of a configuration language, several useful libraries, and an appropriate compiler. Many language features are already available, to provide simplicity as well as efficiency. For example, SNACK implements high-level concepts like routing trees and periodic sensing, making the development of new sensor network applications much easier. Furthermore,

SNACK provides independently implemented services, which can be used by any application. Similarly, Heinzelman *et at.* [113] describe a middleware approach to providing higher-level abstractions of complex, low-level concepts to programmers, thereby easing the design and implementation of applications.

In this chapter, we will review several application examples that employ sensor networks. According to Akyildiz *et al.* [12], we will classify them into the main categories of *habitat and environment monitoring*, *health care*, *home automation*, *smart places*, and *military*.

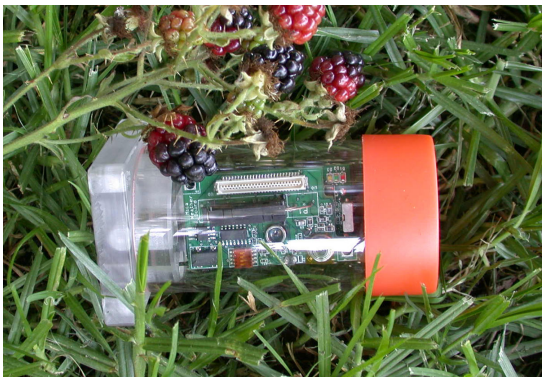
8.2 Habitat Monitoring

In the area of civil applications, habitat monitoring is one of the application drivers for wireless communication technology [49, 232]. Due to their autonomous operation, wireless sensor networks are quite helpful to study many aspects of wildlife. While earlier observation techniques were intrusive, time-consuming, and expensive, deploying an unattended sensor network requires human interaction only to set up, respectively remove the network. Thus, intrusion into the wildlife habitat is reduced substantially. Furthermore, long-term observations become possible, providing live data conveniently.

8.2.1 Great Duck Island

One early project in habitat monitoring was the *Great Duck Island* (GDI) project [165], which was named after a small island in the Gulf of Maine where the project took place. The island is an active site for ecological research, and much of it focuses on the nesting habits of Leach's Storm Petrels [254]. The research objective was to observe the seabirds' behavior and the microclimate in and around the nesting burrows by means of long-term monitoring. The great advantage of using wireless techniques was their non-intrusive and non-disruption characteristics. In the spring of 2002, researchers manually placed 32 sensor nodes in the nesting burrows and installed a base station on the island. Via a satellite link the network was connected to the Internet, offering real-time live data access over the web. The sensors' placement formed a grid across the island from east to west, as well as up the island. Figure 8.1(a) shows an acrylic enclosure that was used to deploy the first generation of nodes above the ground. Later in the project, those nodes were replaced by smaller ones that are illustrated in Figure 8.1(b), which also depicts the placement of nodes within (1) and outside (2) the nesting burrows, the base station (3), and the research station (4) that was connected via satellite (5) to a lab in California.

Before the network was deployed, each node was preconfigured, e. g., a unique address was assigned to each node for routing purposes and identification. At programmed times, each sensor node went on to sent its data to the base station and off to save energy. Data collected by sensor nodes consisted of temperature and humidity readings, barometric pressure, and information about the birds' presence, which was obtained by means of mid-range infrared. Because all nodes were widely scattered over the island, with some nodes being more than 300 meters deep in the forest, data was sent via multi-hop to the base station, by using the nodes' low-power, wireless transceivers. Multiple times, the sensor



(a) Acrylic enclosure

(b) Placement of sensor nodes

Figure 8.1: Sensor nodes used by the Great Duck Island project (from [2])

network was extended by additional sensor nodes. By 2003, over one million readings had been logged and analyzed. In August 2003, the network consisted of about 150 sensor nodes, including more than 25 weather stations.

8.2.2 ZebraNet

In the *ZebraNet* project at Princeton University [128], the behavior of zebras was studied to support wildlife tracking. From the biological point of view, the project's objective was to understand the animals' long-range migration, inter-species interaction, and nocturnal behavior. On the computer systems side, the goal was to design a power-aware and position-aware communication system. In comparison to the GDI project, not the wildlife habitat but the wildlife itself was to be monitored. After one year of research, the first ZebraNet system was deployed at the Mpala Research Centre in central Kenya in January 2004 [274].

In order to track the positions of zebras and monitor their activities, several sensors collecting biometric data like heart rate and temperature were integrated into a collar that could be mounted around the neck of a zebra, as shown in Figure 8.2. Collars also included a GPS device, flash memory for data storage, wireless transceivers, solar cells for battery recharging, and a small CPU. Since there was no cellular service available that covers the entire territory where animals were studied, ad-hoc routing techniques were required. However, because the wireless transceivers were organized in a mobile network, connectivity to a base station could not be guaranteed all the time. Sensor nodes thus operated in a store-and-forward mode: Local data like GPS position samples and biometric data were stored in memory as long as possible. If two zebras were within communication range, they exchanged their data logs with each other using a low-power, short-range radio. To communicate with a mobile base station, which could be a car or a plane, a second, long-distance radio was used. If a base station could be reached by the second radio, all data not already transmitted were delivered. Thus, eventually, almost all data logs could be collected, even though the latency was perhaps high. To avoid sending the same data more than once, each node also held a list of delivered and deleted data that was additionally communicated to peer sensor nodes. Furthermore, data logs were dated by timestamps, providing information as to which data would be erased first if a node ran out of memory.

(a) A dazed zebra

(b) Sensor collar

Figure 8.2: Mounting a sensor collar to the neck of a zebra (from [206])

8.2.3 WildCENSE

The *wildCENSE* project [6] aims at developing a mobile wireless sensor network to monitor the habitat of Indian Nilgai. The Nilgai are antelopes and one of the most commonly seen wild animals of northern India. Since their behavior is similar to that of zebras, wildCENSE relies on the experiences obtained from the ZebraNet project. Animals are collared in the same way to track their geographical area of movement and to monitor their habitat and behavior. The collars comprise similar components, like a microcontroller, GPS, temperature, humidity, light, orientation sensors, off-chip flash memory, an RF XBee-Pro module, a battery, and solar modules for recharging. Again, nodes exchange data logs among each other until they come into the vicinity of a mobile base station.

8.2.4 Cane Toad Monitoring

The *cane toad monitoring* project [120] investigates an *acoustic* wireless sensor network to monitor amphibian populations in the monsoonal woodlands of northern Australia. The project took place in the Kakadu National Park of the Northern Territory, Australia, with the goal to count the population of native frogs and the invasive cane toad species, which is depicted in Figure 8.3(a). Their expanding distribution has raised grave concerns concerning their influence on Australia's fauna, especially in the Kakadu National Park. The dark region in Figure 8.3(b) illustrates their wide-area distribution in 2003.

The system requirements for monitoring cane toads are quite challenging because recognizing vocalizations of different frog species requires high-frequency acoustic sampling, complex signal processing, and a wide-area sensing coverage [223]. The prototype used in the project consists of several less expensive mica2 motes [65] (to achieve a high coverage) and only a few powerful stargate gateways [66]. While acoustic samples are collected by mica2 motes, stargate devices are used to classify

(a) The cane toad

(b) Distribution of cane toads in the Northern Australia in 2003

Figure 8.3: The cane toad and its distribution in Australia (from [120])

different acoustic samples, based on Fast Fourier Transformation (FFT), which transfers acoustic signals from the time domain into the frequency domain. Compression and noise reduction is performed by mica2 motes before transmission of the data to the stargate gateways, in order to reduce the amount of data to be sent. If a sensor node has detected the existence of a frog species, its own location will be used as the location of the frog. Sensor nodes obtain their location either by means of GPS or other localization algorithms. If several adjacent nodes detect a frog species at the same time, the gateway device will coordinate their sampling tasks and calculate the region of overlapping detection areas. Because the long-term migration of frogs is to be tracked, this information is sufficient. By *sampling scheduling*, the efficiency of the system can be increased significantly. Coordinated by the gateway, a schedule for acoustic sampling and transmitting is created. Thus, while one node is compressing and transferring data, another node will be sampling, which increases the processing rate by about 50%. Furthermore, as the transmission of data is coordinated due to sampling scheduling, collisions will mostly be avoided.

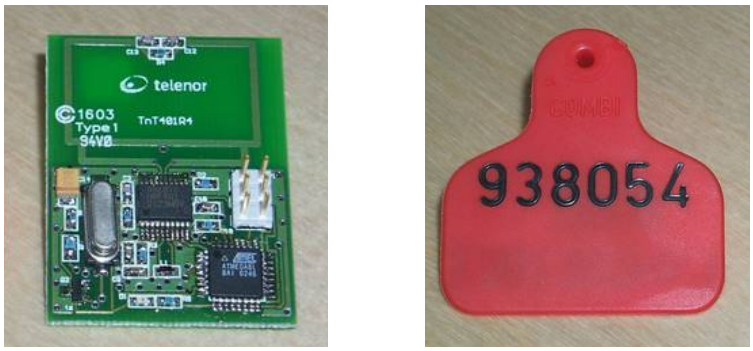
8.2.5 Electronic Shepherd

The *electronic shepherd* project [237] was conducted in Norway with the original aim to develop a system that could be used to keep track of sheep while they are out on grazing land during summertime. Each year, at the beginning of September, the sheep return. But, unfortunately, some sheep are always missing. While some sheep get caught by predators or fall off cliffs, others just remain in the mountains. Hence, keeping track of them would be very helpful.

The challenge was to develop a wireless, mobile communication system that was small enough to be carried by sheep, robust enough to survive in the precipitous terrain, and energy-efficient enough to be able to operate for several months. Furthermore, the location and the "state" of the sheep, e. g., their pulse and temperature, should be able to queried at any time. Because the communication system had to be cheap, it was realized by using low-power and low-bandwidth radio communication equipment, including GPS receivers, UHF radio communication transceivers, and GPRS modems offering a connection to the Internet. Figure 8.4 shows a radio tag used in the electronic shepherd

project. The left picture shows the ordinary version, and the one in the right the packed version that can be attached to the ear of the sheep.

The system is quite innovative as *flock behavior* can be supported. Using the low-cost radios, a flock leader monitors a number of other objects belonging to the same flock. In this way, only the flock leader needs to have a GPRS modem that can be used to reach any computer in the Internet over a cellular GSM or UMTS network. While the electronic shepherd system was designed for the purpose of animal tracking, it can also be used for other applications where objects are to be monitored at a low cost.



(a) Ordinary version (b) Packed version

Figure 8.4: The electronic shepherd radio tag (from [237])

8.3 Environment Observation and Forecast Systems

In contrast to habitat monitoring systems that monitor the behavior of animals or try to track their positions, *environment observation and forecast systems* focus on the observation of the environment itself, aiming to (i) understand biological ecosystems, (ii) alert to emergencies like fires, or (iii) forecast future behaviors or situations. In the following, we will consider four environment monitoring systems. Other examples not mentioned are the *GlacsWeb* project [168] to monitor the behavior of glaciers, *SenSlide* [220] aiming at predicting landslides, the *Four Seasons* project [259] for monitoring the health of human-made structures, or the *Wireless Vineyard* [34].

8.3.1 ALERT

One of the first and widely accepted standards for forecasting potential floods, rainfalls, or tropical cyclones is ALERT, which is an acronym for *automated local evaluation in real-time* [252]. The standard was originally developed by the National Weather Service in the 1970's and has been used by several institutions widely in the U. S., but also in Argentina, Australia, China, India, Indonesia, Jamaica, and Spain. Since that time, many ALERT user groups have cooperated with industrial vendors and government agencies, which are involved in the early detection of flood conditions and their forecasting. However, in addition to flood warnings, the achieved technological advancement of auto-

mated real-time monitoring systems are also useful in many other areas of water resource management and planning.

The ALERT standard specifies a method for using remote sensors in the field, in order to transmit environmental real-time data to a central computer for processing. Usually, ALERT sensor sites are equipped with multiple meteorological sensors, like water level, wind, barometric, and temperature sensors. While there exists a vast number of manufacturers of ALERT hardware, all devices are designed according to common communication criteria. Thus, interchanging software or equipment is often possible. The success of the ALERT technology is especially due to its accuracy, reliability, and low cost, whereby it offers real-time data acquisition, automated hydrologic and hydraulic forecast modeling, as well as automated warnings for many applications.

8.3.2 FireWxNet

In [110], Hartung *et al.* present *FireWxNet*, which is a portable wireless system for monitoring weather conditions in rugged wildland fire environments. With over 96,000 fires burning almost 10 million acres in the U. S. in 2006 alone [181], wildland firefighting has been a necessary but dangerous task long ago. While fire behavior is extremely sensitive to changes in environmental conditions like temperature, relative humidity, and wind, forecasting weather conditions based on previously recorded observations is usually the only way to obtain at least general predictions. Already small changes in elevation may influence the actual fire behavior considerably, making forecasts quite difficult. Accurately monitoring the environmental conditions of fires could thus improve forecasts and warning systems significantly. Furthermore, deploying a wireless network within forest fire regions offers an easy and at the same time safe way to measure and observe local weather conditions over a wide range of locations. While this information was previously unattainable, FireWxNet provides the necessary information in real-time, allowing researchers to better predict fire behavior under safety considerations.

The system was developed as a tiered structure in order to take different deployment capabilities into account. On the upper tier, long-distance communication radios with directional antennas are used to connect weather stations deployed hundreds of kilometers into the wilderness with a base station that provides Internet access via a satellite link. At the other end of the radios, three wireless, multi-hop *weather networks* are connected, which consist of multiple sensor nodes. In addition, two steerable web-enabled cameras are integrated into the network to provide visual verifications of captured weather conditions. Figure 8.5(a) shows a station with a long-distance, directional antenna, connecting a weather network with the base station. Due to lack of electricity, solar panels are used to provide power. Figure 8.5(b) shows a weather station. To protect the sensor nodes from outdoor influences, an enclosure was built around the nodes, which are attached to the ceiling inside. The setup was used during a one week deployment in the Selway-Salmon Complex Fires of 2005.

Closely related to FireWxNet is the *FireBug* application [75], which was developed by researchers at the University of California, Berkeley. During a prescribed burn, they measured environment conditions by means of a sensor network consisting of 10 nodes as a flame front passed by, and demonstrated

(a) A station that connects the weather network to the base station via a directional antenna

(b) Weather station

Figure 8.5: A solar-powered station and a weather station (from [110])

the feasibility of sensor technology in a fire environment. Unlike FireWxNet, it was not intended to forecast fire behavior but rather to measure or track a flame front in itself.

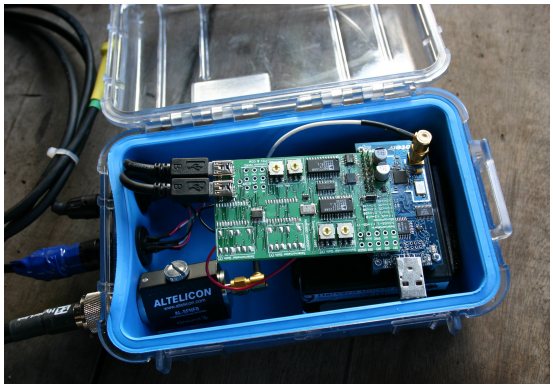
8.3.3 Monitoring Volcanic Eruptions

Researchers at Harvard recently started to study the use of low-power sensor networks for geophysical monitoring. In contrast to traditional data collection equipment, which is often heavy and power-hungry, smaller and lighter sensor nodes offer a new possibility for scientific studies. The advances in microelectronic technology now make deploying hundreds of nodes feasible. However, volcanic studies demand high data rates and fidelity, which need to be addressed in advance.

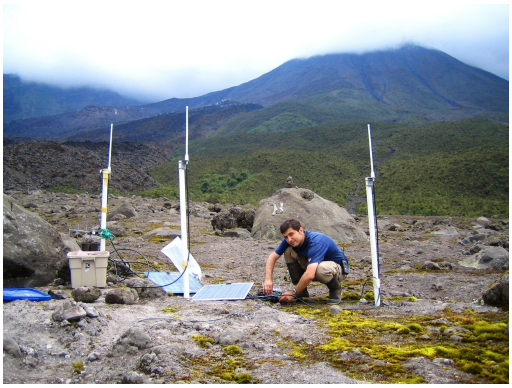
In July 2004, Werner *et al.* deployed a first sensor array at Volcán Tingurahua, an active volcano in central Ecuador, monitoring its seismic activity [248]. By means of low-frequency acoustic (infrasonic) sensors, volcanic eruptions were monitored over a period of 54 hours, in which at least nine explosions occurred. After collection of infrasonic signals, data were transmitted to a remote base station, using a wireless link about 9 km long. Data were collected by means of triggered event detection and reliable data retrieval. The distributed event detector automatically triggered the transmission of data if correlated signals were received by multiple nodes. Bandwidth usage, as well as energy consumption, could thus be reduced considerably. In order to achieve high data fidelities, all sensor nodes were time-synchronized using separate GPS receivers. Finally, the collected data were compared to data acquired at a nearby *wired* sensor network for verification.

While the deployment in 2004 was a first proof of concept, a larger network was deployed over 3 km on the Reventator volcano in the western Amazon in Ecuador in August 2005 [249]. The network consisted of 16 nodes that were equipped with a microphone and a seismometer to collect acoustic and seismic data. Via a multi-hop network, data was relayed to a gateway node, which used a long-distance radio for communicating with a base station. While most nodes were connected to the gateway over three or fewer hops, some nodes needed even more than six hops. Using a GPS receiver along with a multi-hop time-synchronization protocol, a network-wide global time was established. Figure 8.6(a) shows a wireless sensor node together with an attached interface board used to connect the external

antenna and sensors. While seismometers were buried nearby the nodes, microphones were mounted on PVC poles and shielded from wind and elements with plastic tapes. As shown in Figure 8.6(b), those poles were used to elevate antennas so as to minimize any ground effects, which may have reduced the radio communication range.



(a) A sensor node with attached interface hardware



(b) External antennas mounted on PVC poles

Figure 8.6: Monitoring equipment used on the Reventator volcano (from [247])

Currently, much larger node arrays are deployed to monitor volcanic eruptions over several months. Continuous Internet connectivity via a satellite uplink is also provided, allowing real-time access to sensed data. Another idea which was recently realized was to *trigger* a satellite to take pictures of active volcanos after an eruption. Researchers from several institutes are collaborating within the *Volcano SensorWeb* project at NASA's Jet Propulsion Laboratory [71].

8.3.4 Redwood Ecophysiology

In the *redwood ecophysiology*[1] project [239], the microclimate of coastal redwood canopies is being studied. While a lot of research has already been done, much of the work focused on the microclimate either on the ground or above tree canopies. However, the microclimate is also affected by (and itself affects) the interactions of trees and their environment. Hence, the dynamic processes *within* trees also need to be understood. Due to the lack of empirical data, the physiology of the entire tree canopy is still an open problem. The goal of the redwood project thus was to study the ecophysiology of redwood trees by monitoring the microclimate *between* the ground and the canopy. By using a large number of wireless sensor nodes (weather stations), such environmental dynamics as spatial variation, as well as temporal microclimate dynamics were captured. Over 44 days, the life of 70-meter tall redwood trees was monitored on the Sonoma Coast, California. Every five minutes, data was collected by about 40 to 50 nodes per tree and transmitted to more powerful *data loggers* that stored data in local databases and then transmitted these to an offsite database over GPRS cellular modems.

The final placement of nodes in the redwood trees is schematically shown in Figure 8.7. Nodes were manually placed at different elevations, spaced roughly two meters apart. Different types of sensors

[1] *Ecophysiology*, or *environmental physiology*, is a biological discipline which studies the adaptation of physiology to environmental conditions [253].

were used in order to measure air temperature, relative humidity, light conditions, and photosynthetically active solar radiation.

Although the deployment and the data collection process were quite successful, it turned out that long-term environmental monitoring should also include a network monitoring component to account for data inconsistencies and node failures. Thus, monitoring the system's performance will become important as well, in order to provide real-time information and alert researchers in case of anomalies.

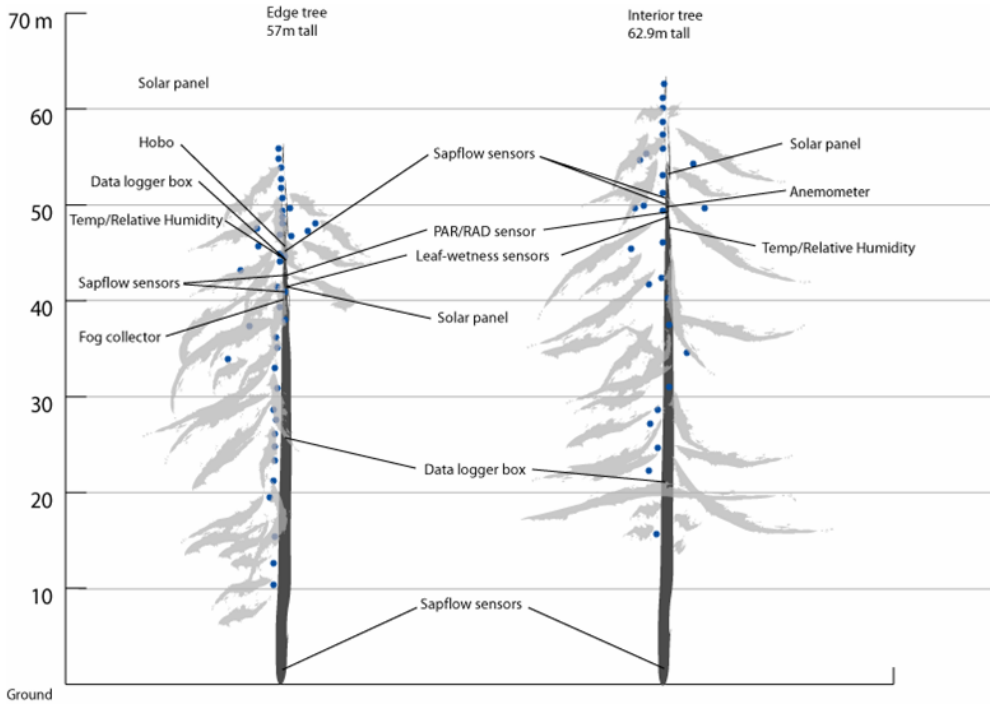


Figure 8.7: The placement of nodes in redwood trees (from [68])

8.4 Health Care

Sensor networks can also help in several areas of health care. For example, health-care applications involve the monitoring of human physiological data, drug administration in hospitals, the tracking of patients and doctors, or recognition of emergency situations [11]. While today many sensors are already used in hospitals to monitor the vital functions of patients, they are often connected through *wired* systems. The mobility of patients is thus very restricted. Furthermore, vital functions are monitored only if necessary. Hence, using *wireless* sensors instead can improve the patients' quality of life substantially, and more rapidly detect any kind of emergency situation. Patients can be under permanent monitoring while they are inside the hospital or at home [19]. If there is any change in the state of a patient, alerts can be triggered automatically, providing all the necessary information. Another application can be the tracking of doctors or patients within a hospital, but also the tracking of expensive medical equipment [20].

8.4.1 Smart Home Care

Researchers at Stanford, California, have proposed a wireless, smart home care system to account for the needs of the elderly or persons who need medical assistance [233]. Motivated by the growing domain of caregiving, they developed a wireless sensor network that supports multiple sensing and event detection modalities based on sensor fusion and distributed vision-based reasoning. Image sensing is employed to verify, as well as to analyze, data reported by other sensors. For example, in the event of an accidental fall, the fall will be detected by a wireless badge node carried by the person under care [109], alerting the network's base station. Automatically, the caregiving center will be called and a voice connection to the person will be established. In monitoring the indoor environment, the voice channel is carried over an IEEE 802.15.4 radio link, which is also used to track the position of the person based on signal strength measurements. Also triggered by the detection of a fall, wall-mounted image sensor nodes will provide additional information on the user's condition, which is quite useful for the subsequent emergency service.

8.4.2 Implanted Biomedical Devices

Another challenging application area is considered by Schwiebert *et al.* in [216], who describe their experiences with *biomedical devices* that operate within the human body to compensate diseases. They are currently working on an *artificial retina* as part of the *smart sensors and integrated microsystems* (SSIM) project. The goal of the project is to restore vision to visually-impaired and blind individuals. The artificial retina should be permanently implanted as shown in Figure 8.8 in the eye and provide sufficient visual functionality so as to "see" at an acceptable level. The developed retina prosthesis consists of about 100 micro-sensors, organized as an array within a *smart sensor*, which is connected to a wireless RF transceiver. Acting as the base station for the artificial retina, an external computer system is used for image processing as well as for translating between images recorded by a built-in camera and signals intended to be transmitted to the retina. Therefore, special networking protocols needed to be developed to account for the challenges of human-embedded smart sensor arrays.

Implanted biomedical devices can also be used in other applications, including *glucose level monitors*, *organ monitors*, *cancer detectors*, or *general health monitors*. While implanting biomedical sensors within the human body may have the potential to revolutionize medicine, special requirements first need to be accomplished. Besides the limitations of power and computational capabilities, the design of biomedical devices must be bio-compatible, fault-tolerant, energy-efficient, and scalable. Furthermore, the wireless system must be reliable, almost maintenance-free, and ultra-safe, emphasizing application-specific solutions that differ vastly from other sensor network application domains.

8.5 Home Automation and Smart Places

As another category of applications for sensor networks, the field of *home automation and smart places* offers context-aware assistance to people. The vision is that all electronic appliances will form a network and collaborate to satisfy the needs of a user and afford a high degree of convenience [12].

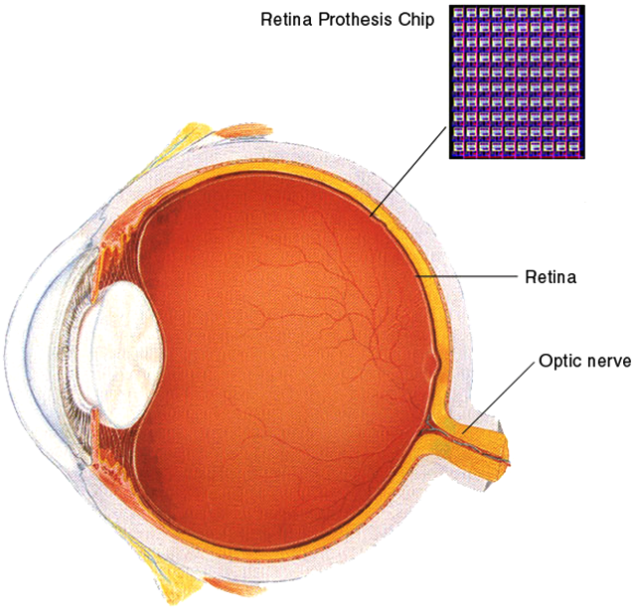


Figure 8.8: Location of the retina prosthesis chip within the eye (from [216])

For example, networking almost everything opens new possibilities to control and monitor special devices in order to trigger predefined events, to account for security and safety issues, to reduce energy costs, or simply to increase convenience. However, today's home automation solutions are highly proprietary and often restricted to special infrastructures, like non-standardized networks or power supply cables [189].

8.5.1 The Intelligent Home

Intelligent Home [1] is a Western Australian company which provides fully integrated solutions for home automation. Figure 8.9 illustrates their showroom, which is used to present the components of a smart home [147]. Their solutions cover *structured cabling*, *multi-room audio*, *controlled lighting*, *security and CCTV*, *phone/intercom systems*, and *home theater*. The system is under the total control of a user, anywhere and anytime. The multi-room audio system allows a user to listen to his music in any room, without the need of multiple stereo systems. Over the installed cable system, each room is connected to the main stereo system, allowing a user to change titles, sound, and volume. By the use of touch-screen control panels, any audio source such as a tuner, CD or DVD can be distributed to designated locations around the home. The controlled lighting system makes it feasible to program lighting to control any light anywhere in the home. Predefined configurations for any situation are possible, e. g., switching on particular lights upon coming home or leaving, creating different lighting levels in different rooms, or providing security features by programming specific lights to switch on at night. Security and safety is also provided by CCTV. By placing cameras in specific areas around the home, e. g., over the front door or in the backyard, these areas are kept under surveillance and can be monitored from any TV within the home. In combination with the phone system, this provides efficient communication inside as well as outside the home. Entertainment is offered by the home

Figure 8.9: The intelligent home showroom (from [1])

theater center, which includes various audio and video components, allowing a user to watch movies, listen to music, or play games.

8.5.2 MavHome

Though the intelligent home just described offers multiple possibilities to program and control connected devices and complex tasks, the inhabitants' *way of living* is not considered by the system automatically. Concerning this matter, the goal of the *MavHome* project [3] is to create a home that *acts* like an intelligent agent. Through sensors, MavHome perceives the state of a home and its inhabitants and acts appropriately. The system must have the ability to predict and reason about different situations. For example, consider the following scenario from [62]: "*At 6:45am, MavHome turns up the heat because it has learned that the home needs 15 minutes to warm to optimal waking temperature. The alarm sounds at 7:00, after which the bedroom light and kitchen coffee maker turn on. Bob steps into the bathroom and turns on the light. MavHome records this interaction, displays the morning news on the bathroom video screen, and turns on the shower. When Bob finishes grooming, the bathroom light turns off while the kitchen light and display turn on, and the news program moves to the kitchen screen. During breakfast, Bob requests the janitor robot to clean the house. When Bob leaves for work, MavHome secures the home, and starts the lawn sprinklers despite knowing the 30% predicted chance of rain. Because the refrigerator is low on milk and cheese, MavHome places a grocery order. When Bob arrives home, his grocery order has arrived and the hot tub is waiting for him.*"

Implementing such an intelligent agent system requires that knowledge about the environment as well as inhabitants can be acquired and applied. By means of effective prediction algorithms, MavHome is able to learn about the inhabitants' behavior and to predict their next actions. Previously seen interactions between inhabitants and various devices are used for this purpose. In [70], Cook *et al.* present two algorithms for predicting actions inhabitants will take and for learning a policy to control

the home. Both algorithms rely on techniques known from *artificial intelligence* and trade-off the overall goal of minimizing the prediction error and maximizing comfort and efficiency.

8.5.3 Embedded Script-Driven Home Automation

One possibility of making the installation and configuration of home automation more convenient is to use adaptable plug-and-play solutions based on generic sensor networks as proposed in [105, 106]. In contrast to proprietary solutions that are usually restricted, *wireless* solutions do not rely on any infrastructures and thus offer a higher degree of freedom. Especially, if the installation of a system is carried out at a future date which could not be considered during the construction of a building.

The idea of the prototype described in [105] is to gather all kinds of sensor readings in a home and forward them hop-by-hop to an embedded system, which is referred as the home automation server. Therefore, the sensor nodes have to be distributed in the house according to the requirements of the considered applications. During this distribution process, the user is given hints by the system on where to place intermediate nodes for the purpose of communication. After that, the operational phase takes place, in which events are forwarded by the network to the root node and eventually to the home automation server in a tree-like fashion. Each time a new event is detected, the server runs over a list of so-called script statements which can be defined by the user via a web-interface as shown in Figure 8.10. In case of a match between the received event and the matching part of a statement, one or more actions are performed which can either be executed by the sensor nodes themselves or by multiple plugs which can be controlled via an Ethernet connection by the embedded home automation server itself. For example, the action may serve one particular purpose like, e. g., baby surveillance. In this case, the executed action could be as simple as signaling the user with the buzzer or by sending him some information over the web by the embedded server, e.g., to submit an SMS via an external service.

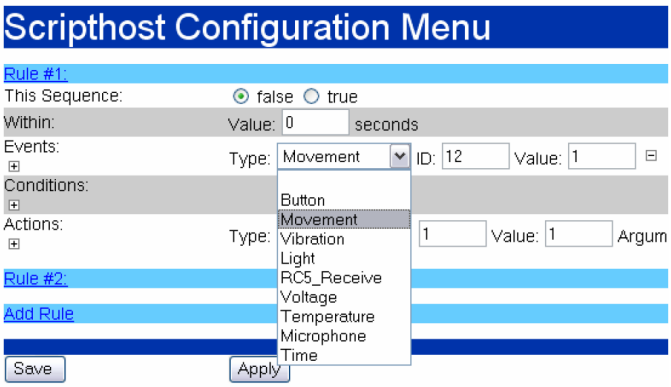


Figure 8.10: Browser-based configuration of home automation rules

The user can define an arbitrary number of rules, each of which appears as a single line that can be unfolded to a dialog for later editing. A rule consists of three elements whereas the triggering event and the action to be performed are the two compulsory elements. Whenever an event like a sensor reading occurs, it is compared with all rules. If the event matches a rule, the according action is performed. Whether an action is performed must not always depend on an event only but also on

a condition which, unlike the event, persists over some time and does not occur at a single moment only. So a rule can have an arbitrary number of conditions which have to be met in addition to an occurring event. A condition can e.g., be a specific time of the day or a prior sensor reading like light or temperature. Multiple events, conditions and even actions can be defined within a single rule by the user. Thus, the strength and contribution of the application lies in the combination of a larger number of sensor readings, which allows to derive higher level semantics as compared to reacting on single sensor readings only.

8.6 Military Applications

Initially, research and development in sensor networks have actually been driven by defense and military applications, focusing, e. g., on enemy tracking, battlefield surveillance, detection of chemical, biological, and nuclear attacks, or reconnaissance of opposing forces [58]. Due to their self-organization techniques, large-scale deployment, autonomous operation, and increased fault tolerance, wireless sensor networks are very well suited for such surveillance missions, which are often driven by target tracking and classification [31]. Especially unmanned surveillance missions have become of great interest for the military.

The *distributed sensor networks* (DSN) program, which was started around 1980 and funded by the *Defense Advanced Research Projects Agency* (DARPA) [236], was one of the first modern research projects in the field of sensor networks. The technology components for a DSN ranged from acoustic sensors, high-level protocols for communication [226], processing techniques and algorithms, distributed software, to signal processing and situation assessment [250]. For demonstration purposes, distributed acoustic tracking was chosen as the target problem [58].

8.6.1 Sensor Information Technology

Even though the DSN program was intended to address networks consisting of a large number of sensor nodes, the appropriate components were not yet technically mature. Leveraging the latest technology advances, DARPA launched a new research program named *sensor information technology* (SensIT) in 1999. Based on *micro-electro-mechanical systems* (MEMS) [92], inexpensive and low-power processors are offered, which allow for large deployments of wireless sensor networks. The goals of the SensIT project were to develop new networking techniques that could be used in unstructured and sometimes hostile environments, and to develop networked information processing procedures [139]. Thus, the main function of SensIT is to detect, identify, locate, and track any kind of objects.

In [174], Meesookho *et al.* describe experimental results for vehicle target classifications in battlefields. The aim was to classify military vehicles passing through a field of a large number of sensor nodes between two checkpoints, as shown in Figure 8.11. Each sensor node collected acoustic and seismic data from vehicles in order to locate and classify ten different armored vehicles. Two neighboring nodes located along the track were selected as data sources for the experiment, performing

Figure 8.11: Overview of the SensIT scenario (from [173])

collaborative signal processing and data fusion. Together with the sensor and vehicle locations, time-stamped details of the experiment were logged for later processing. The experimental results showed that a 50% relative improvement in the classification error could be achieved by means of collaboration. Only when the convoy contained a larger number of vehicles or vehicles of various types, was the performance degraded. However, the collaboration between two nodes still yielded significant improvements.

8.6.2 EnviroTrack

He *et al.* present a similar system for the object tracking of hostile targets in [111]. Acquiring and verifying information about enemy capabilities and the positions of moving targets is performed by energy-efficient surveillance based on the *EnviroTrack* middleware [8]. For evaluation purposes, a network consisting of 70 mica2 motes [65, 115] equipped with dual-axis magnetometers was used. As typical surveillance missions last from a few days to several months, an energy-aware design scheme was required because it may not be possible to replenish the energy of power-constraint sensor devices manually due to inaccessible hostile territories. The main issues of the EnviroTrack systems thus were longevity, adjustable sensitivity, stealthiness, and effectiveness. Adjustable sensitivity of sensors makes it possible to adapt to different surveillance terrains and security requirements. While critical missions may require a high degree of sensitivity in order to capture all potential targets, decreasing the system's sensitivity may be useful if false alarms and unnecessary power dissipation are to be minimized. Achieving stealthiness is also crucial, especially for military applications. As RF signals can be intercepted easily, a zero communication exposure is desired in the absence of significant events. Thereby, the effectiveness of the system may be slightly relaxed if the accuracy of location estimates and the latency of detection reports are still acceptable. In so doing, the system may trade off energy-awareness and surveillance performance. To detect and track the positions of hostile targets, it is also required that all nodes be time-synchronized [44, 89] and know their own positions [60, 186].

8.6.3 Counter-Sniper System

The counter-sniper system *PinPtr* [224] tackles the problem of locating snipers in urban environments, and tries to overcome the limitations of existing systems. The system's performance is analyzed by means of real measurements, obtained at a U.S. Army facility. Based on a wireless ad-hoc sensor network, shooters are detected and located quite accurately, achieving an average location error of about one meter within two seconds. Figure 8.12 shows the graphical user interface of the system. The estimated position of the shooter is shown as the red circle, while the direction of the shot is indicated by an arrow. Coordinates and elevations are displayed in the top right corner. The location of sensor nodes are shown as green circles.

Figure 8.12: Graphical user interface of PinPtr (from [224])

The system consists of a large number of mica2 sensor motes [65], which provide good coverage of the urban environment and high accuracy. Furthermore, due to a high degree of redundancy, even multiple sensor failures can be tolerated. Via acoustic signals like muzzle blasts and shockwaves, sensors are able to detect a shot and measure its time of arrival. Afterwards, all timestamps are delivered to a central base station, using data aggregation and an underlying gradient-based converge-cast routing system. Assigning one node as the root, each node rebroadcasts data packets up to three times along multiple paths towards the root node. However, while data reporting is thus fast and robust, the message overhead is significant.

Using a similar tree structure, all sensor nodes synchronize their local clocks to the clock of a selected root node. Because precise time synchronization is crucial for the application in order to compute the sniper's location from the nodes' time of arrival values, the global time is estimated by synchronizing nodes with nodes one level higher. Therefore, the *flooding time synchronization protocol* [167] is used. Since data reports additionally need to include the positions of sensor nodes, Simon *et al.* propose a self-localization algorithm based on acoustic range estimates [207] and passive acoustic sensor localization [140]. However, due to practical shortcomings, the current counter-sniper system uses sensors at known locations.

8.7 Conclusions

In this chapter, we have considered several examples of different sensor network application areas, ranging from habitat monitoring, environment observation and forecasting to health care, home automation, smart places, and military applications. However, there is no claim to completeness due to the vast number of proposed and employed applications existing today. We have seen that depending on the application and the deployment conditions, the network requirements may be quite different. For example, some scenarios demand mobility or store-and-forward concepts, while others rely on densely populated, but static networks, transmitting sensed data immediately. Hence, protocols and algorithms developed are usually targeted to specific application scenarios.

The algorithms and protocols proposed in this thesis were tailored to static networks, as they can be found in several application scenarios. Due to their energy limitations, these networks require an energy-efficient design in order to operate a long time in an unattended manner. Furthermore, as the examples in this chapter have shown, data often need to be forwarded to a sink node for later processing or storage. Thus, the approaches proposed in the previous chapters may be used for multiple real-world deployments, as they are already in use.

In the future, the technology advances will allow further applications that are not possible today. As sensor nodes become smaller, networks tend to be deployed more densely. Thus, sometime, the vision of *smart dust* introduced by Pister in 2001 [245] will no longer merely be science fiction. Sensor motes the size of a grain of sand or even of dust particles may become reality and form the basis for integrated, massively distributed sensor networks. However, energy efficiency will remain one of the key challenges that need to be addressed.

According to Culler's keynote speech in 2006 [69], sensor networks will additionally become the *next tier* of the Internet. While we are today able to essentially connect *everybody*, we will be able to connect and observe essentially *everything* of value in the future. Sensor networks will become ubiquitous and will be integrated seamlessly, networking most of the physical world. Achieving these visions remains a great challenge, for research, as well as for the industry.

CHAPTER 9

Conclusions and Future Work

"Every now and then, go away, have a little relaxation, for when you come back to your work your judgment will be surer. Go some distance away because then the work appears smaller and more of it can be taken in at a glance, and a lack of harmony and proportion is more readily seen."

– L. Da Vinci –

9.1 Conclusions

In the past few years, wireless sensor networks have attracted a great deal of interest. Several applications have either newly emerged or have been simplified by means of small-sized, wireless sensor nodes that are able to remain unattended in a specific environment. Communication over a wireless medium entails a simple deployment of hundreds (and soon maybe thousands) of nodes placed almost anywhere. Since sensor nodes are equipped with a central processing unit, application tasks can be distributed throughout the network easily, making the network more scalable. Moreover, localized algorithms, which rely only on information a node has received from its neighborhood, enable the use of simple instructions that even low-power sensor nodes are able to execute.

The limitations in terms of processing, storage, and energy capacity require new algorithms and protocols, which take the specific conditions of a wireless sensor network into account. In this thesis, our main focus was on energy consumption, which is heavily influenced by the use of radio communication. Reducing the energy consumption extends the overall lifetime of the network and is thus an important goal that implemented algorithms and protocols must fulfill. Furthermore, it is commonly assumed that the sensor nodes are battery-powered, and batteries are not replaceable, either due to inaccessible terrain or to the use of low-cost hardware, making the battery replacement inefficient.

However, solely minimizing the energy consumption in the network is not possible because the network operation should not (or almost not) be affected. We accounted for this trade-off by focusing

instead on the *energy efficiency*, which was defined as the number of data bytes delivered to a sink node per consumed energy unit. In this way, both the delivery process for data issued by sensor nodes, as well as the energy consumption within the network could be optimized at the same time.

All algorithms and protocols proposed in this thesis were implemented and evaluated by means of real hardware, the *embedded sensor board* (ESB), which was developed as a research platform by the Free University of Berlin. We first described the platform in detail and investigated the radio characteristics of its TR1001 transceiver. As the TR1001 module is only able to send single bits over a radio frequency, the transceiver is connected to a *universal asynchronous receiver/transmitter* (UART) circuit. The UART makes it possible to transmit an entire byte stream according to a predefined bit rate. Because the communication is asynchronous, data bytes are framed by means of start and stop bits before they are transmitted. As long as no bit errors occur and the state machines of the sender and receiver are synchronized, the communication works well. However, once the synchronization is lost, the remaining transmissions may become useless if no resync is performed. In this case, the receiver may misinterpret data bits as start or stop bits and vice versa. This behavior is extremely prejudicial to *forward error correction* (FEC) because the number of bit errors may increase substantially. Thus, we proposed an appropriate *resync mechanism* which is able to re-synchronize the sender's and receiver's state machines, rendering the remaining communication still useful if FEC is employed [37]. We considered different types of FEC codes: a single and double error correcting code, a Hamming code, a Reed-Solomon code, a random linear fountain code, and a Raptor code. By means of real-world experiments, we showed that the double error correcting code and the Reed-Solomon code performed similarly and that both codes are suitable to correct bit errors within a packet [43]. If entire packets within a stream are to be recovered (as would be desirable for bulk data transmissions), using random linear fountain codes will be most suitable, as the redundancy of the code need not be predefined [41].

While forward error correction is able to reduce the number of packets retransmitted by a sending node and thus reduces the energy consumption in the network, much energy may still be wasted if packets are forwarded along inefficient routes. We demonstrated by means of simulations that forwarding strategies based on a minimum hop counter or a maximum end-to-end packet delivery ratio may perform badly if they are used in a wireless network. Considering a static network, we proposed the use of an *energy-efficient strategy* (EEF) in order to maximize the average number of data bytes per consumed energy unit along a forwarding path [36, 42]. We showed that the performance of EEF in terms of energy efficiency is superior to that of a broad range of other approaches proposed in the literature. In addition, we proposed the concept of *multi-link* forwarding (MEEF) that takes advantage of opportunistic routing: Rather than sending a data packet to a single receiver, the packet is sent to several nodes at once, from among which a forwarder is selected afterwards. In so doing, the energy efficiency could be further improved because packet retransmissions in case of packet loss could often be avoided. However, even if multi-link forwarding was not applied, single-link energy efficient forwarding (SEEF) still outperformed all other strategies considered. SEEF and MEEF were carefully simulated and additionally implemented on the ESB platform. The results showed that again both strategies outperformed other approaches, even in our real-world experiments.

Although the EEF strategy achieved the best trade-off between the number of data packets delivered and the energy required to forward packets, the energy consumption within the network may not be balanced because nodes along energy-efficient forwarding paths may be used more often than others.

Thus, if additionally the time at which the first node runs out of energy should be maximized, the residual energy of nodes needs to be taken into account as well. For this purpose, we presented a strategy called *lifetime-efficient forwarding* (LEF), which extended EEF appropriately [35, 38]. Similar to EEF, we proposed two versions (SLEF and MLEF) to account for single-link and multi-link forwarding. Again, both strategies were fully analyzed, simulated, and implemented. The results showed that LEF outperformed EEF clearly in terms of lifetime efficiency. However, avoiding low-energy nodes came at the expense of energy efficiency and increased the overall energy consumption of the network. Thus, in real-world deployments, it is more likely that the type of application will determine whether EEF or LEF is preferable.

Exploiting the content of data packets is another way to reduce the number of packet transmissions, and thus the energy consumption. By means of in-network processing, correlated data may be combined and aggregated to a single packet that can then be forwarded on its own, instead of sending each packet individually. While such data aggregation can be used independently of the forwarding strategy, we pointed out that considering the effects of aggregation during the construction of forwarding paths is extremely beneficial. If it is known *a priori* which nodes will issue data packets, building an aggregation tree that is tailored to energy efficiency is desirable. For this purpose, we proposed *single-link* and *multi-link energy-efficient aggregation forwarding* (SEEAF and MEEAF) [40]. Both strategies yielded forwarding paths with a high potential for aggregation, rendering the forwarding process much more efficient. We presented a mechanism to avoid forwarding cycles, which, if they are not taken into account, may otherwise occur during the construction of an aggregation tree. The mechanism can be used by EEF and LEF, too, if the packet reception rates on forwarding links start to change. Simulation results, as well as the experimental results obtained by means of our real-world testbed, showed that EEAF clearly outperformed EEF in terms of the information delivery ratio, energy consumption, and energy efficiency. Compared to a centralized solution, the performance of EEAF was only slightly worse and thus quite good, keeping in mind that EEAF is a fully distributed algorithm.

Besides those forwarding algorithms, we studied the impact of a topology management algorithm to reduce the energy consumption of idle listening. Because keeping the radio transceivers of *all* nodes on all the time is a huge waste of energy, we proposed a *topology and energy control algorithm* (TECA) to establish a backbone of only *active* nodes [39]. While active nodes keep their radios on, non-active nodes go into a low-power sleep mode with their communication radios turned off. We decided that maintaining connectivity within the network is an important objective of many applications and thus considered it as a key design issue. At first, TECA divides the network into several clusters of nodes and selects for each cluster a cluster head. By calculating a spanning tree that covers all cluster heads, interconnecting bridge nodes are then selected, which afterwards become part of the backbone topology. All other nodes are considered redundant and will turn off their radios for a certain period of time to save energy. By using only local information, TECA operates in a fully distributed manner and outperforms other approaches clearly. Results from both simulations and real-world experiments have identified its superior performance in terms of the number of active nodes, energy consumption, network connectivity, and network lifetime.

9.2 Future Work

In future research, the issues discussed in this thesis can be extended. For example, it would be interesting to investigate the synchronization problem on other hardware than that of the ESB platform. As the next generation of the ESB does not employ any UART circuits for wireless communication, it is expected that the transmission of a bit stream will be much more reliable. However, since data bytes are no longer framed by start and stop bytes, the time synchronization between a sender and a receiver need to be more precise. What would happen in this case if a receiver got out of sync could be a topic of future work.

Furthermore, the analyses of FEC could be extended. Other codes than the ones considered in this thesis could be implemented and investigated concerning their suitability for sensor networks. In this context, particularly *adaptive* FEC would be of great interest, since it allows dynamic tuning of the amount of redundancy per packet. The dissemination of data could be further analyzed as well. Other protocols than the acknowledgement-based and the request-based protocol could be considered and evaluated for different FEC codes.

The focus in designing the EEF algorithm was solely on energy efficiency. Other functions that could be incorporated are the end-to-end packet latency, which may be important with respect to real-time traffic, and congestion control. Both metrics could be measured on-the-fly during packet transmissions and taken into account when forwarding paths are established. Moreover, the selection of appropriate metrics for different applications needs to be investigated. Extending EEF to the multi-sink case can be achieved by calculating the node's energy efficiency for each sink separatively. In case data packets may be forwarded to any sink, only the best one in terms of energy efficiency need to be stored. Otherwise, the energy-efficient path of each sink is stored individually. Data packets are then sent to that sink that requested them. Thus, the EEF algorithm does not need many changes, as the number of sinks only influences its computation time and memory usage.

Other extensions of EEF are also possible: Note that for the sake of simplicity, we assumed for our mathematical analyses that packets received are not buffered by forwarding nodes. That is, it is assumed that some packets may be forwarded more than once, e. g., if acknowledgements get lost. Considering appropriate receiving buffers makes the mathematical analysis more difficult because then two cases need to be distinguished, namely whether or not a packet has already been forwarded. However, the mathematical calculation of the end-to-end energy efficiency would be more accurate. Furthermore, a lighter solution than the polling mechanism used for multi-link forwarding could be considered: Instead of actively polling each potential forwarder individually, each forwarding node could set a timer upon receiving a packet, based on its position in the forwarder list. If a timer expires, an acknowledgement could be sent back immediately to inform the sender about the successful transmission. Upon receiving an acknowledgement, the sender would then need to broadcast a "stop" message to avoid the transmissions of other acknowledgements and to trigger the actual forwarding of the packet. Whether or not such an approach could increase the energy efficiency of the protocol remains to be discussed.

The integration of EEF with existing MAC protocols like S-MAC or Wise-MAC could also be part of future work, especially, if the employed MAC requires the use of an energy model other than that

assumed by EEF. As several MAC protocols already rely on periodic beacon packets, they could easily be augmented by EEF information in order to establish forwarding paths. Moreover, MAC-related information regarding contention could be integrated into EEF.

In order to balance the energy consumption in the network and to avoid the burn-out of frequently used sensor nodes, we proposed the LEF algorithm. However, as shown by means of simulations and experiments, this comes at the expense of less energy efficiency. Thus, an adaptive forwarding behavior would be desirable. It may also be possible to incorporate application knowledge to distinguish between simple relay nodes and more important sensing nodes. Similar to EEF, the EEAF strategy could then be augmented by a lifetime component, too. While relay nodes could only perform forwarding, more powerful sensor nodes could perform aggregation. To achieve a longer lifetime of such nodes, they could propagate their residual energy levels to their neighborhood. In this way, they would only be used if really necessary. Analyzing the impact of partially correlated data could also be part of future research. Moreover, the simulations could be augmented by means of data obtained from real deployments, rather than assuming that all data issued by nodes can be aggregated without any extra space.

The topology management algorithm TECA can be extended in different ways. So far, only the residual energy of nodes is used for clustering the network; the nodes with the most energy become cluster heads, which are connected by bridge nodes afterwards. Other metrics like the number of neighbors or special hardware characteristics could be taken into account as well. Moreover, cluster heads could be augmented by specific controlling tasks, e. g., by providing a *time division multiple access* within their cluster. The impact of variable radio transmission powers could be investigated, too. Highly interesting in this context would be how to find the best transmission power in terms of energy efficiency. At last, investigating how TECA, EEF, LEF, and EEAF would perform in conjunction is also very interesting.

Although we designed all algorithms to be independent of geographic information, taking advantage of geographic knowledge (if available) is certainly possible. Appropriate approaches that combine geographic routing and energy-efficient forwarding have already been proposed in the literature. Especially when the network is not static but mobile, it is very useful to exploit such information.

The experiments carried out for all algorithms offered several insights, but more comprehensive experiments are required to obtain results that are better validated. First of all, the number of sensor nodes would need to be increased, at least to about 100, in order to provide enough forwarding alternatives and possibilities to cluster the network. By means of real application scenarios, all nodes could then be deployed, e. g, within a building, and used over a longer period of time. Achieving an optimal deployment of nodes would be desirable and thus interesting from a practical point of view. In addition, analyzing the integration of different protocols remains an important research task, which needs to be tackled in the future.

Bibliography

[1] *Automated Solutions*. URL http://www.intelligenthome.com.au/. Online. Accessed at 2007-06-23.

[2] *The Great Duck Island Project*. URL http://www.greatduckisland.net/. Online. Accessed at 2007-06-23.

[3] *MavHome - Managing an Adaptive Versatile Home*. URL http://cygnus.uta.edu/mavhome/. Online. Accessed at 2007-06-23.

[4] *The Official Bluetooth Web Site*. URL http://www.bluetooth.com/bluetooth/. Online. Accessed at 2007-06-23.

[5] *ScatterWeb*. URL http://cst.mi.fu-berlin.de/projects/ScatterWeb/. Online. Accessed at 2007-06-23.

[6] *Sensor Networks for Wildlife Monitoring*. URL http://intranet.da-iict.org/~ranjan/research/papers/wildcense/index.htm. Online. Accessed at 2007-06-23.

[7] *21 Ideas for the 21st Century*. In: Business Week, pages 78–167, August 1999.

[8] Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, S., Stankovic, J., Stoleru, R., and Wood, A. *EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks*. In: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS), pages 582–589. IEEE Computer Society, Tokyo, Japan, March 2004.

[9] Ahn, J.-S., Hong, S.-W., and Heidemann, J. *An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks*. In: Journal of Communications and Networks, Vol. 7, No. 4, pages 489–499, December 2005.

[10] Akkaya, K. and Younis, M. F. *A Survey on Routing Protocols for Wireless Sensor Networks*. In: Ad Hoc Networks, Vol. 3, No. 3, pages 325–349, May 2005.

[11] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. *A Survey on Sensor Networks*. In: IEEE Communications Magazine, Vol. 40, No. 8, pages 102–114, August 2002.

[12] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. *Wireless Sensor Networks: A Survey*. In: Computer Networks, Vol. 38, No. 4, pages 393–422, March 2002.

[13] Apilo, O., Lassila, P., and Virtamo, J. *Performance of Local Forwarding Methods for Geographic Routing in Large Ad Hoc Networks*. In: Proceedings of the 5th IFIP Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET), pages 121–128. Lipari, Italy, June 2006.

[14] Arampatzis, T., Lygeros, J., and Manesis, S. *A Survey of Applications of Wireless Sensors and Wireless Sensor Networks*. In: Proceedings of the 13th Mediterranean Conference on Control and Automation (MED), pages 719–724. IEEE Computer Society, Cyprus, Greek, July 2005.

[15] Avin, C. and Krishnamachari, B. *The Power of Choice in Random Walks: An Empirical Study*. In: Proceedings of the 9th ACM International Simposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWIM), pages 219–228. ACM Press, Torremolinos, Spain, October 2006.

[16] Awerbuch, B. *Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems*. In: Proceedings of the 19th ACM Symposium on Theory of Computing (STOC), pages 230–240. ACM Press, New York, NY, USA, 1987.

[17] Awerbuch, B. and Leighton, T. *A Simple Local-Control Approximation Algorithm for Multicommodity Flow*. In: Proceedings of the 34th IEEE Symposium on Foundations of Computer Science (FOCS), pages 459–468. IEEE Computer Society, Palo Alto, CA, USA, November 1993.

[18] Awerbuch, B. and Leighton, T. *Improved Approximation Algorithms for the Multi-Commodity Flow Problem and Local Competitive Routing in Dynamic Networks*. In: Proceedings of the 26th ACM Symposium on Theory of Computing (STOC), pages 487–496. ACM Press, Montreal, Canada, May 1994.

[19] Baker, C. R., Armijo, K., Belka, S., Benhabib, M., Bhargava, V., Burkhart, N., Minassians, A. D., Dervisoglu, G., Gutnik, L., Haick, M. B., Ho, C., Koplow, M., Mangold, J., Robinson, S., Rosa, M., Schwartz, M., Sims, C., Stoffregen, H., Waterbury, A., Leland, E. S., Pering, T., and Wright, P. K. *Wireless Sensor Networks for Home Health Care*. In: Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications (AINA), pages 832–837. IEEE Computer Society, Niagara Falls, Canada, May 2007.

[20] Baldus, H., Klabunde, K., and Müsch, G. *Reliable Set-Up of Medical Body-Sensor Networks*. In: Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN), pages 353–363. Springer, Berlin, Germany, January 2004.

[21] Bao, L. and Garcia-Luna-Aceves, J. J. *Topology Management in Ad Hoc Networks*. In: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), pages 129–140. ACM Press, Annapolis, MD, USA, June 2003.

[22] Berlekamp, E. R. Algebraic Coding Theory. McGraw Hill, New York, NY, USA, 1968.

[23] Bhardwaj, M., Garnett, T., and Chandrakasan, A. P. *Upper Bounds on the Lifetime of Sensor Networks*. In: Proceedings of the 5th IEEE International Conference on Communications (ICC), pages 785–790. IEEE Computer Society, Helsinki, Finland, June 2001.

[24] Bierl, L. Das große MSP430 Praxisbuch. Franzis Verlag GmbH, Poing, Germany, 2004. In German.

[25] Biswas, S. and Morris, R. *Opportunistic Routing in Multi-Hop Wireless Networks*. In: ACM SIGCOMM Computer Communication Review, Vol. 34, No. 1, pages 69–74, January 2004.

[26] Biswas, S. and Morris, R. *ExOR: Opportunistic Routing in Multi-Hop Wireless Networks*. In: Proceedings of the 28th ACM International SIGCOMM. ACM Press, Philadelphia, PA, USA, August 2005.

[27] Blough, D., Leoncini, M., Resta, G., and Santi, P. *The K-Neigh Protocol for Symmetric Topology Control in Ad Hoc Networks*. In: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), pages 141–152. ACM Press, Annapolis, MD, USA, June 2003.

[28] Blough, D. and Paolo, S. *Investigating Upper Bounds on Network Lifetime Extension for Cell-Based Energy Conservation Techniques in Stationary Ad Hoc Networks*. In: Proceedings of the 8th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 183–192. ACM Press, Atlanta, GA, USA, September 2002.

[29] Boukerche, A., Cheng, X., and Linus, J. *Energy-Aware Data-Centric Routing in Microsensor Networks*. In: Proceedings of the 6th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWIM), pages 42–49. ACM Press, San Diego, CA, USA, September 2003.

[30] Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. *Gossip Algorithms: Design, Analysis and Applications*. In: Proceedings of the 24th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1653–1664. IEEE Computer Society, Miami, FL, USA, March 2005.

[31] Brooks, R. R., Ramanathan, P., and Sayeed, A. M. *Distributed Target Classification and Tracking in Sensor Networks*. In: Proceedings of the IEEE, Vol. 91, No. 8, pages 1163–1171, August 2003.

[32] Bulusu, N., Heidemann, J., and Estrin, D. *GPS-Less Low Cost Outdoor Localization for Wireless Sensor Networks*. In: Personal Communications Magazine, Vol. 7, No. 5, pages 28–34, October 2000.

[33] Burkhart, M., von Rickenbach, P., Wattenhofer, R., and Zollinger, A. *Does Topology Control Reduce Interference?* pages 9–19. ACM Press, Roppongi Hills, Tokyo, Japan, May 2004.

[34] Burrell, J., Brooke, T., and Beckwith, R. *Vineyard Computing: Sensor Networks in Agricultural Production*. In: IEEE Pervasive Computing, Vol. 3, No. 1, pages 38–45, January–March 2004.

[35] Busse, M., Haenselmann, T., and Effelsberg, W. *A Comparison of Lifetime-Efficient Forwarding Strategies for Wireless Sensor Networks*. In: Proceedings of the 3rd ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN), pages 33–40. ACM Press, Torremolinos, Spain, October 2006.

[36] Busse, M., Haenselmann, T., and Effelsberg, W. *Energy-Efficient Forwarding Schemes for Wireless Sensor Networks*. In: Proceedings of the 7th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM), pages 125–133. IEEE Computer Society, Niagara-Falls, Buffalo-NY, USA, June 2006.

[37] Busse, M., Haenselmann, T., and Effelsberg, W. *The Impact of Resync on Wireless Sensor Network Performance*. In: Proceedings of the 1st Workshop on Performance Control in Wireless Sensor Networks (PWSN), pages 63–70. Coimbra, Portugal, May 2006.

[38] Busse, M., Haenselmann, T., and Effelsberg, W. *Poster Abstract: A Lifetime-Efficient Forwarding Strategy for Wireless Sensor Networks*. In: Adjunct Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN), pages 20–21. ETH Zurich, Zurich, Switzerland, February 2006.

[39] Busse, M., Haenselmann, T., and Effelsberg, W. *TECA: A Topology and Energy Control Algorithm for Wireless Sensor Networks*. In: Proceedings of the 9th ACM International Simposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWIM), pages 317–321. ACM Press, Torremolinos, Spain, October 2006.

[40] Busse, M., Haenselmann, T., and Effelsberg, W. *Energy-Efficient Aggregation Forwarding for Wireless Sensor Networks*. In: Proceedings of the IARIA International Conference on Sensor Technologies and Applications (SENSORCOMM), pages 584–589. IEEE Computer Society, Valencia, Spain, October 2007.

[41] Busse, M., Haenselmann, T., and Effelsberg, W. *Energy-Efficient Data Dissemination for Wireless Sensor Networks*. In: Proceedings of the 3rd IEEE International Workshop on Sensor Networks and Systems for Pervasive Computing (PERSENS), pages 301–306. IEEE Computer Society, White Plains, NY, USA, March 2007.

[42] Busse, M., Haenselmann, T., and Effelsberg, W. *Energy-Efficient Forwarding in Wireless Sensor Networks*. In: Pervasive and Mobile Computing, 2008. To appear.

[43] Busse, M., Haenselmann, T., King, T., and Effelsberg, W. *The Impact of Forward Error Correction on Wireless Sensor Network Performance*. In: Proceedings of the 2nd ACM Workshop on Real-Word Wireless Sensor Networks (REALWSN), pages 67–71. ACM Press, Uppsala, Sweden, June 2006.

[44] Busse, M. and Streichert, T. *Time Synchronization*. In: Wagner, D. and Wattenhofer, R., editors, Algorithms for Sensor and Ad Hoc Networks, pages 359–380. Springer, Berlin, Germany, 2007.

[45] Byers, J., Luby, M., Mitzenmacher, M., and Rege, A. *A Digital Fountain Approach to Reliable Distribution of Bulk Data*. In: Proceedings of the 21st ACM International SIGCOMM, pages 56–67. ACM Press, Vancouver, BC, Canada, September 1998.

[46] Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraula, G., and E.Bonabeau. Self-Organization in Biological Systems. Princeton University Press, Princeton, NJ, USA, 2003.

[47] Cao, Q., He, T., Fang, L., Abdelzaher, T., Stankovic, J., and Son, S. *Efficiency Centric Communication Model for Wireless Sensor Networks*. In: Proceedings of the 25th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1–12. IEEE Computer Society, Barcelona, Spain, April 2006.

[48] Cerpa, A., Busek, N., and Estrin, D. *SCALE: A Tool for Simple Connectivity Assessment in Lossy Environments*. Technical Report 21, Center for Embedded Networked Sensing, University of California, Los Angeles, CA, USA, September 2003.

[49] Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., and Zhao, J. *Habitat Monitoring: Application Driver for Wireless Communications Technology*. In: SIGCOMM Computer Communication Review, Vol. 31, No. 2, pages 20–41, April 2001.

[50] Cerpa, A. and Estrin, D. *ASCENT: Adaptive Self-Configuring Sensor Network Topologies*. In: Proceedings of the 21st Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1567–1576. IEEE Computer Society, New York, NY, USA, June 2002.

[51] Cerpa, A., Wong, J. L., Kuang, L., Potkonjak, M., and Estrin, D. *Statistical Model of Lossy Links in Wireless Sensor Networks*. In: Proceedings of the 4th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), pages 81–88. IEEE Computer Society, Los Angeles, CA, USA, April 2005.

[52] Chang, J. and Tassiulas, L. *Maximum Lifetime Routing in Wireless Sensor Networks*. In: IEEE/ACM Transactions on Networking, Vol. 12, No. 4, pages 609–619, August 2004.

[53] Chang, J.-H. and Tassiulas, L. *Fast Approximate Algorithms for Maximum Lifetime Routing in Wireless Ad-Hoc Networks*. In: Proceedings of the IFIP-TC6 / European Commission International Conference (NETWORKING), pages 702–713. Springer, Paris, France, May 2000.

[54] Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. *Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks*. In: Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 85–96. ACM Press, Rome, Italy, July 2001.

[55] Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. *Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks*. In: Wireless Networks, Vol. 8, No. 5, pages 481–494, September 2002.

[56] Chen, J.-Y., Pandurangan, G., and Xu, D. *Robust Computation of Aggregates in Wireless Sensor Networks: Distributed Randomized Algorithms and Analysis*. In: Proceedings of the 4th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), pages 348–355. IEEE Press, Los Angeles, CA, USA, April 2005.

[57] Chien, R. *Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes*. In: IEEE Transactions on Information Theory, Vol. 10, No. 4, pages 357–363, October 1964.

[58] Chong, C.-Y. and Kumar, S. P. *Sensor Networks: Evolution, Opportunities, and Challenges*. In: Proceedings of the IEEE, Vol. 91, No. 8, pages 1247–1256, August 2003.

[59] Choudhury, R. R. and Va, N. H. *MAC-Layer Anycasting in Ad Hoc Networks*. In: ACM SIGCOMM Computer Communication Review, Vol. 34, No. 1, pages 75–80, January 2004.

[60] Chu, H.-C. and Jan, R.-H. *A GPS-Less Self-Positioning Method for Sensor Networks*. In: Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pages 629–633. IEEE Computer Society, Fuduoka, Japan, July 2005.

[61] Ciciriello, P., Mottola, L., and Picco, G. P. *Efficient Routing from Multiple Sources to Multiple Sinks in Wireless Sensor Networks*. In: Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN), pages 34–50. Springer, Delft, The Netherlands, January 2007.

[62] Cook, D. J., Youngblood, M., Heierman, E. O., Gopalratnam, K., Rao, S., Litvin, A., and Khawaja, F. *MavHome: An Agent-Based Smart Home*. In: Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM), pages 521–524. IEEE Computer Society, Fort Worth, TX, USA, March 2003.

[63] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Introduction to Algorithms. MIT Press, Cambridge, MA, USA, 2001.

[64] Couto, D. S. J. D., Aguayo, D., Chambers, B., and Morris, R. *A High-Throughput Path Metric for Multi-Hop Wireless Routing*. In: Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 134–146. ACM Press, San Diego, CA, USA, September 2003.

[65] Crossbow. The MICA2 Mote Platform. URL http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf. Online. Accessed at 2007-06-23.

[66] Crossbow. The Stargate Gateway Platform. URL http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf. Online. Accessed at 2007-06-23.

[67] Culler, D., Estrin, D., and Srivastava, M. B. *Overview of Sensor Networks*. In: IEEE Computer, Vol. 37, No. 8, pages 41–49, August 2004.

[68] Culler, D. E. *Toward the Sensor Network Macroscope*. Keynote at the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), May 2005. URL http://www.cs.berkeley.edu/~culler/talks/mobihoc.ppt. Online. Accessed at 2007-06-23.

[69] Culler, D. E. *The Next Tier of the Internet*. Keynote at the 3rd IEEE International Conference on Broadband Communications, Networks, and Systems (BROADNETS), October 2006. URL http://www.cs.berkeley.edu/~culler/talks/. Online. Accessed at 2007-06-23.

[70] Das, S. and Cook, D. J. *Designing Smart Environments: A Paradigm Based on Learning and Prediction*. In: Shorey, R., Ananda, A. L., Chan, M. C., and Ooi, W. T., editors, Mobile, Wireless, and Sensor Networks, pages 337–357. J. Wiley & Sons, New York, NY, USA, March 2006.

[71] Davies, A., Chien, S., Wright, R., Miklius, A., Kyle, P., Welsh, M., Johnson, J., Tran, D., Schaffer, S., and Sherwood, R. *Sensor Web Enables Rapid Response to Volcanic Activity*. In: EOS Transactions American Geophysical Union, Vol. 87, page 1, January 2006.

[72] De Couto, D. S. J., Aguayo, D., Chambers, B., and Morris, R. *Performance of Multihop Wireless Networks: Shortest Path is Not Enough*. In: ACM SIGCOMM Computer Communication Review, Vol. 33, No. 1, pages 83–88, January 2003.

[73] Dimakis, A. G., Sarwate, A. D., and Wainwright, M. J. *Geographic Gossip: Efficient Aggregation for Sensor Networks*. In: Proceedings of the 5th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), pages 69–76. ACM Press, Nashville, TN, USA, April 2006.

[74] Ding, M., Cheng, X., and Xue, G. *Aggregation Tree Construction in Sensor Networks*. In: Proceedings of the 58th IEEE International Vehicular Technology Conference (VTC), pages 2168–2172. IEEE Press, Orlando, FL, USA, October 2003.

[75] Doolin, D. and Sitar, N. *Wireless Sensors for Wildfire Monitoring*. In: SPIE International Symposium on Smart Structures and Materials, pages 477–484. San Diego, CA, USA, May 2005.

[76] Doshi, S., Bhandare, S., and Brown, T. X. *An On-Demand Minimum Energy Routing Protocol for a Wireless Ad Hoc Network*. In: ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 6, No. 3, pages 50–66, July 2002.

[77] Dousse, O., Mannersalo, P., and Thiran, P. *Latency of Wireless Sensor Networks with Uncoordinated Power Saving Mechanisms*. In: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), pages 109–120. ACM Press, Roppongi Hills, Tokyo, Japan, May 2004.

[78] Dubois-Ferrière, H., Estrin, D., and Vetterli, M. *Packet Combining in Sensor Networks*. In: Proceedings of the 3rd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 102–115. ACM Press, San Diego, CA, USA, November 2005.

[79] Dulman, S., Nieberg, T., Wu, J., and Havinga, P. *Trade-Off Between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks*. pages 1918–1922. ACM Press, New Orleans, LA, USA, March 2003.

[80] Egorova-Förster, A. and Murphy, A. L. *A Feedback-Enhanced Learning Approach for Routing in WSN*. In: Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN), pages 397–408. VDE, Bern, Switzerland, February 2007.

[81] El-Hoiydi, A. and Decotignie, J.-D. *WiseMAC: An Ultra Low Power MAC Protocol for the Downlink of Infrastructure Wireless Sensor Networks*. In: Proceedings of the 9th IEEE International Symposium on Computers and Communications (ISCC), pages 244–251. IEEE Computer Society, Alexandria, Egypt, June 2004.

[82] ElBatt, T. A., Krishnamurthy, S. V., Connors, D., and Dao, S. K. *Power Management for Throughput Enhancement in Wireless Ad-Hoc Networks*. In: Proceedings of the 4th IEEE International Conference on Communications (ICC), pages 1506–1513. IEEE Computer Society, New Orleans, LA, USA, June 2000.

[83] Elson, J., Girod, L., and Estrin, D. *Fine-Grained Network Time Synchronization Using Reference Broadcasts*. In: SIGOPS Operating System Review, Vol. 36, No. SI, pages 147–163, December 2002.

[84] Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. *Next Century Challenges: Scalable Coordination in Sensor Networks*. In: Proceedings of the 5th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 263–270. ACM Press, Seattle, WA, USA, August 1999.

[85] Forney, G. D. *On Decoding BCH Codes*. In: IEEE Transactions on Information Theory, Vol. 11, No. 4, pages 549–557, October 1965.

[86] Gallager, R. G., Humblet, P. A., and Spira, P. *A Distributed Algorithm for Minimum Weight Spanning Trees*. In: ACM Transactions on Programming Languages and Systems, Vol. 5, No. 1, pages 66–77, January 1983.

[87] Gan, L., Liu, J., and Jin, X. *Agent-Based, Energy-Efficient Routing in Sensor Networks*. In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 472–479. IEEE Computer Society, New York, NY, USA, July 2004.

[88] Ganeriwal, S., Ganesana, D., Shim, H., Tsiatsis, V., and Srivastava, M. B. *Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks*. In: Proceedings of the 3rd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 130–141. ACM Press, San Diego, CA, USA, November 2005.

[89] Ganeriwal, S., Kumar, R., and Srivastava, M. B. *Timing-Sync Protocol for Sensor Networks*. In: Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 138–149. ACM Press, Los Angeles, CA, USA, 2003.

[90] Ganesan, D., Govindan, R., Shenker, S., and Estrin, D. *Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks*. In: ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 5, No. 4, pages 11–25, October 2001.

[91] Garay, J. A., Kutten, S., and Peleg, D. *A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees*. In: SIAM Journal of Computing, Vol. 27, No. 1, pages 302–316, February 1998.

[92] Gardner, J. W., Varadan, V. K., and Awadelkarim, O. O. Microsensors, MEMS and Smart Devices. J. Wiley & Sons, New York, NY, USA, November 2001.

[93] Garey, M. R. and Johnson, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, NY, USA, 1979.

[94] Garg, N. and Konemann, J. *Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems*. In: Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), pages 300–309. IEEE Computer Society, Palo Alto, CA, USA, November 1998.

[95] Geier, J. Wireless LANs. SAMS, Indianapolis, IN, USA, 2001.

[96] Ghosh, S., Basu, S., and Touba, N. A. *Reducing Power Consumption in Memory ECC Checkers*. In: Proceedings of the 4th IEEE International Test Conference (ITC), pages 1322–1331. IEEE Computer Society, October 2004.

[97] Gnawali, O., Yarvis, M., Heidemann, J., and Govindan, R. *Interaction of Retransmission, Blacklisting, and Routing Metrics for Reliability in Sensor Network Routing*. In: Proceedings of the 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), pages 34–43. IEEE Computer Society, Santa Clara, CA, USA, October 2004.

[98] Godfrey, P. B. and Ratajczak, D. *Naps: Scalable, Robust Topology Management in Wireless Ad Hoc Networks*. In: Proceedings of the 3rd ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), pages 443–451. ACM Press, Berkeley, CA, USA, April 2004.

[99] Goldenberg, D. K., Bihler, P., Yang, Y. R., Cao, M., Fang, J., Morse, A. S., and Anderson, B. D. O. *Localization in Sparse Networks using Sweeps*. In: Proceedings of the 12th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 110–121. ACM Press, Los Angeles, CA, USA, September 2006.

[100] Gomez, J., Campbell, A. Z., Naghshineh, M., and Bisdikian, C. *Conserving Transmission Power in Wireless Ad Hoc Networks*. In: Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP), pages 24–34. IEEE Computer Society, Riverside, CA, USA, November 2001.

[101] Greenstein, B., Kohler, E., and Estrin, D. *A Sensor Network Application Construction Kit (SNACK)*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 69–80. ACM Press, Baltimore, MD, USA, November 2004.

[102] Gulliver, T. A. and Bhargava, V. K. *A Systematic (16,8) Code for Correcting Double Errors and Detecting Triple-Adjacent Errors*. In: IEEE Transactions on Computers, Vol. 42, No. 1, pages 109–112, January 1993.

[103] Gupta, H., Zhou, Z., Das, S. R., and Gu, Q. *Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution*. In: IEEE/ACM Transactions on Networking, Vol. 14, No. 1, pages 55–67, February 2006.

[104] Hac, A. Wireless Sensor Network Designs. J. Wiley & Sons, New York, NY, USA, 2004.

[105] Haenselmann, T., Busse, M., King, T., Effelsberg, W., and Fuchs, M. *Embedded Script-Driven Home-Automation with Sensor Networks*. In: Proceedings of the 1st IFIP Home Networking Conference. Paris, France, December 2007. To Appear.

[106] Haenselmann, T., King, T., Effelsberg, W., Busse, M., and Fuchs, M. *Skriptbasierte drahtlose Gebudeautomation mit Sensornetzen*. In: PIK – Praxis der Informationsverarbeitung und Kommunikation: Fachzeitschrift fr den Einsatz von Informationssystemen, Vol. 03, pages 163–169, July 2007. In German.

[107] Hamming, R. W. *Error Detection and Error Correction Codes*. In: The Bell System Technical Journal, Vol. 26, No. 2, pages 147–160, March 1950.

[108] Han, K.-H., Ko, Y.-B., and Kim, J.-H. *A Novel Gradient Approach for Efficient Data Dissemination in Wireless Sensor Networks*. In: Proceedings of the 60th IEEE International Vehicular Technology Conference (VTC), pages 2979–2983. IEEE Press, Los Angeles, CA, USA, September 2004.

[109] Hansen, T. R., Eklund, J. M., Sprinkle, J., Bajcsy, R., and Sastry, S. *Using Smart Sensors and a Camera Phone to Detect and Verify the Fall of Elderly Persons*. In: Proceedings of the 3rd European Medicine, Biology and Engineering Conference (EMBEC), pages 2486–2489. Prague, Czech Republic, November 2005.

[110] Hartung, C., Han, R., Seielstad, C., and Holbrook, S. *FireWxNet: A Multi-Tiered Portable Wireless System for Monitoring Weather Conditions in Wildland Fire Environments*. In: Proceedings of the 4th ACM International Conference on Mobile Systems, Applications, and Services (MOBISYS), pages 28–41. ACM Press, Uppsala, Sweden, June 2006.

[111] He, T., Krishnamurthy, S., Stankovic, J. A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Hui, J., and Krogh, B. *Energy-Efficient Surveillance System Using Wireless Sensor Networks*. In: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications, and Services (MOBISYS), pages 270–283. ACM Press, Bosten, MA, USA, June 2004.

[112] Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. *Building Efficient Wireless Sensor Networks with Low-Level Naming*. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), pages 146–159. ACM Press, Banff, Alberta, Canada, October 2001.

[113] Heinzelman, W. B., Murphy, A. L., Carvalho, H. S., and Perillo, M. A. *Middleware to Support Sensor Network Applications*. In: IEEE Network, Vol. 18, No. 1, pages 6–14, January/February 2004.

[114] Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. *Energy-Efficient Communication Protocol for Wireless Microsensor Networks*. In: Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences (HICSS), pages 1–10. IEEE Computer Society, Maui, HI, USA, January 2000.

[115] Hill, J. and Culler, D. *Mica: A Wireless Platform for Deeply Embedded Networks*. In: IEEE Mirco, Vol. 22, No. 6, pages 12–24, November 2002.

[116] Hof, H.-J. *Applications of Sensor Networks*. In: Wagner, D. and Wattenhofer, R., editors, Algorithms for Sensor and Ad Hoc Networks, pages 1–20. Springer, Berlin, Germany, 2007.

[117] Hsiao, M. Y. *A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes*. In: IBM Journal of Research and Development, Vol. 14, No. 4, pages 395–401, July 1970.

[118] Hu, L. *Topology Control for Multihop Packet Radio Networks*. In: IEEE Transactions on Communications, Vol. 41, No. 10, pages 1474–1481, October 1993.

[119] Hu, L. and Evans, D. *Localization for Mobile Sensor Networks*. In: Proceedings of the 10th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 45–57. ACM Press, Philadelphia, PA, USA, October 2004.

[120] Hu, W., Tran, V. N., Bulusu, N., Chou, C.-T., Jha, S., and Taylor, A. *The Design and Evaluation of a Hybrid Sensor Network for Cane-Toad Monitoring*. In: Proceedings of the 4th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), pages 503–508. IEEE Press, Los Angeles, CA, USA, April 2005.

[121] Hui, J. W. and Culler, D. *The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 81–94. ACM Press, Baltimore, MD, USA, November 2004.

[122] Intanagonwiwat, C., Estrin, D., Govindan, R., and Heidemann, J. *Impact of Network Density on Data Aggregation in Wireless Sensor Networks*. In: Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS), pages 457–458. IEEE Press, Vienna, Austria, July 2002.

[123] Intanagonwiwat, C., Govindan, R., and Estrin, D. *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks*. In: Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 56–67. ACM Press, Boston, MA, USA, August 2000.

[124] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. *Directed Diffusion for Wireless Sensor Networking*. In: IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pages 2–16, February 2003.

[125] Jia, L., Noubir, G., Rajaraman, R., and Sundaram, R. *GIST: Group-Independent Spanning Tree for Data Aggregation in Dense Sensor Networks*. In: Proceedings of the 2nd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), pages 282–304. Springer, San Francisco, CA, USA, June 2006.

[126] Johnson, D. B. and Maltz, D. A. *Dynamic Source Routing in Ad Hoc Wireless Networks*. In: Imielinski, T. and Korth, H., editors, Mobile Computing, Vol. 353, pages 153–181. Kluwer Academic Publishers, 1996.

[127] Jones, C. E., Sivalingam, K. M., Agrawal, P., and Chen, J.-C. *A Survey of Energy-Efficient Network Protocols for Wireless Networks*. In: Wireless Networks, Vol. 7, No. 4, pages 343–358, August 2001.

[128] Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D. *Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet*. In: ACM SIGOPS Operating Systems Review, Vol. 36, No. 5, pages 96–107, December 2002.

[129] Kalpakis, K., Dasgupta, K., and Namjoshi, P. *Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks*. In: Computer Networks, Vol. 42, No. 6, pages 697–716, August 2003.

[130] Karl, H. Protocols and Architectures for Wireless Sensor Networks. Springer, Berlin, Germany, 2005.

[131] Kim, J. and Bohacek, S. *A Comparison of Opportunistic and Deterministic Forwarding in Mobile Multihop Wireless Networks*. In: Proceedings of the 1st ACM International Workshop on Mobile Opportunistic Networking (MOBIOPP), pages 9–16. ACM Press, San Juan, Puerto Rico, June 2007.

[132] Köppe, E., Liers, A., Ritter, H., and Schiller, J. *Low-Power Image Transmission in Wireless Sensor Networks Using ScatterWeb Technology*. In: Proceedings of the 1st IEEE International Workshop on Broadband Advanced Sensor Networks (BASENETS), pages 69–76. IEEE Computer Society, San Josè, CA, USA, October 2004.

[133] Kottapalli, V. A., Kiremidjian, A. S., Lynch, J. P., Carryer, E., Kenny, T. W., Law, K. H., and Lei, Y. *Two-Tiered Wireless Sensor Network Architecture for Structural Health Monitoring*. In: Proceedings of the SPIE International Symposium on Smart Structures and Materials, pages 9–18. San Diego, CA, US, March 2003.

[134] Kou, L., Markowsky, G., and Berman, L. *A Fast Algorithm for Steiner Trees*. In: Acta Informatica, Vol. 15, No. 2, pages 141–145, June 1981.

[135] Krishnamachari, B., Estrin, D., and Wicker, S. *Impact of Data Aggregation in Wireless Sensor Networks*. In: Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS), pages 575–578. IEEE Press, Vienna, Austria, July 2002.

[136] Kruskal, J. B. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. In: Proceedings of the American Mathematical Society, Vol. 7, pages 48–50, 1956.

[137] Kulkarni, P., Ganesan, D., Shenoy, P., and Lu, Q. *SensEye: A Multi-Tier Camera Sensor Network*. pages 229–238. ACM Press, Hilton, Singapore, November 2005.

[138] Kulkarni, S. S. and Wang, L. *MNP: Multihop Network Reprogramming Service for Sensor Networks*. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS), pages 7–16. IEEE Press, Columbus, OH, USA, June 2005.

[139] Kumar, S. and Shepherd, D. *SensIT: Sensor Information Technology for the Warfighter*. In: Proceedings of the 4th ISIF International Conference on Information Fusion (FUSION), pages TuC–1–3–TuC–1–9. Montreal, Canada, August 2001.

[140] Kushwaha, M., Molnar, K., Salla, J., Volgyesi, P., Maróti, M., and Lédeczi, A. *Sensor Node Localization Using Mobile Acoustic Beacons*. In: Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), pages 1–9. IEEE Computer Society, Washington, DC, USA, November 2005.

[141] Lacan, J. and Perennou, T. *Evaluation of Error Control Mechanisms for 802.11b Multicast Transmissions*. In: Proceedings of the 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WIOPT), pages 1–6. IEEE Computer Society, Boston, MA, USA, April 2006.

[142] Larsson, P. *Selection Diversity Forwarding in a Multihop Packet Radio Network with Fading Channel and Capture*. In: ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 5, No. 4, pages 47–54, October 2001.

[143] Lee, M. and Wong, V. W. S. *An Energy-Efficient Spanning Tree Algorithm for Data Aggregation in Wireless Sensor Networks*. In: Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), pages 300–303. IEEE Press, Victoria, BC, Canada, August 2005.

[144] Lee, M. and Wong, V. W. S. *LPT for Data Aggregation in Wireless Sensor Networks*. In: Proceedings of the 48th IEEE Global Telecommunications Conference (GLOBECOM), pages 2969–2974. IEEE Communication Society, St. Louis, MO, USA, November 2005.

[145] Lee, M. and Wong, V. W. S. *E-Span and LPT for Data Aggregation in Wireless Sensor Networks*. In: Computer Communications, Vol. 29, No. 13–14, pages 2506–2520, March 2006.

[146] Lee, S., Bhattacharjee, B., and Banerjee, S. *Efficient Geographic Routing in Multihop Wireless Networks*. pages 230–241. ACM Press, Urbana-Champaign, IL, USA, May 2005.

[147] Lesser, V., Atighetchi, M., Benyo, B., Horling, B., Raja, A., Vincent, R., Wagner, T., Xuan, P., and Zhang, S. *The Intelligent Home Testbed*. In: In Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop). Seattle, WA, USA, June 1999.

[148] Lesser, V., Ortiz, C. L., and Tambe, M. Distributed Sensor Networks: A Multiagent Perspective. Kluwer Acedemic Publisher, Boston, MA, USA, 2003.

[149] Levis, P., Patel, N., and Culler, D. *Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks*. In: Proceedings of the 1st USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI), pages 15–28. ACM Press, San Francisco, CA, USA, March 2004.

[150] Li, N. and Hou, J. C. *Topology Control in Heterogeneous Wireless Networks: Problems and Solutions*. In: Proceedings of the 23rd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 243–254. IEEE Computer Society, Hong Kong, China, March 2004.

[151] Li, N., Hou, J. C., and Sha, L. *Design and Analysis of an MST-Based Topology Control Algorithm*. In: Proceedings of the 22nd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1702–1712. IEEE Computer Society, San Francisco, CA, USA, April 2003.

[152] Li, N., Hou, J. C., and Sha, L. *Design and Analysis of an MST-Based Topology Control Algorithm*. In: IEEE Transactions on Wireless Communications, Vol. 4, No. 3, pages 1195–1206, May 2005.

[153] Li, Q., Aslam, J., and Rus, D. *Online Power-Aware Routing in Wireless Ad-Hoc Networks*. In: Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 97–107. ACM Press, Rome, Italy, July 2001.

[154] Li, X.-Y. *Algorithmic, Geometric and Graphs Issues in Wireless Networks*. In: Wireless Communications and Mobile Computing, Vol. 3, No. 2, pages 119–140, March 2003.

[155] Lin, S. and Costello, D. J. Error Control Coding: Fundamentals and Applications. Prentice Hall, 1983.

[156] Liu, H., Ma, H., Zarki, M. E., and Gupta, S. *Error Control Schemes for Networks: An Overview*. In: Mobile Networks and Applications, Vol. 2, No. 2, pages 167–182, September 1992.

[157] Lloyd, E. L., Liu, R., Marathe, M. V., Ramanathan, R., and Ravi, S. S. *Algorithmic Aspects of Topology Control Problems for Ad Hoc Networks*. In: Mobile Networks and Applications, Vol. 10, No. 1–2, pages 19–34, February 2005.

[158] Luby, M. *LT Codes*. In: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS), pages 271–282. IEEE Computer Society, Vancouver, BC, Canada, November 2002.

[159] Luby, M., Mitzenmacher, M., Shokrollahi, A., and Spielman, D. *Efficient Erasure Correcting Codes*. In: IEEE Transactions on Information Theory, Vol. 47, No. 2, pages 569–584, February 2001.

[160] Lucent Technologies, Inc. URL `http://www.lucent.com/`. Online. Accessed at 2007-06-23.

[161] Lundgren, H., Nordstro, E., and Tschudin, C. *Coping with Communication Gray Zones in IEEE 802.11b-Based Ad Hoc Networks*. In: Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WOWMOM), pages 49–55. ACM Press, Atlanta, GA, USA, September 2002.

[162] Luo, H., Ye, F., Cheng, J., Lu, S., and Zhang, L. *A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks*. In: Wireless Networks, Vol. 11, No. 1–2, pages 161–175, January 2005.

[163] MacKay, D. J. C. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, Cambridge, UK, 2003.

[164] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. *TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks*. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), pages 131–146. Boston, MA, USA, December 2002.

[165] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. *Wireless Sensor Networks for Habitat Monitoring*. In: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), pages 88–97. ACM Press, Atlanta, GA, USA, September 2002.

[166] Marina, M. and Das, S. *On-Demand Multipath Distance-Vector Routing in Ad Hoc Networks*. In: Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP), pages 14–23. IEEE Computer Society, Mission Inn, CA, USA, November 2001.

[167] Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. *The Flooding Time Synchronization Protocol*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 39–49. ACM Press, Baltimore, MD, USA, November 2004.

[168] Martinez, K., Riddoch, A., Hart, J., and Ong, R. *A Sensor Network for Glaciers*. In: Stevenson, A. and Wright, S., editors, Intelligent Spaces, pages 125–138. Springer, 2006.

[169] Massey, J. L. *Shift-Register Synthesis and BCH Decoding*. In: IEEE Transactions on Information Theory, Vol. 15, No. 1, pages 122–127, January 1969.

[170] Mauve, M., Widmer, J., and Hartenstein, H. *A Survey on Position-Based Routing in Mobile Ad-Hoc Networks*. In: IEEE Network Magazine, Vol. 15, No. 6, pages 30–39, November 2001.

[171] McAuley, A. J. *Reliable Broadband Communications Using a Burst Erasure Correcting Code*. In: Proceedings of the 13th ACM International SIGCOMM, pages 287–306. ACM Press, Philadelphia, PA, USA, September 1990.

[172] McMahan, M. L. *Evolving Cellular Handset Architectures but a Continuing, Insatiable Desire for DSP MIPSVector*. In: Texas Instruments Technical Journal, Vol. 17, No. 1, pages 1–10, March 2000.

[173] Meesookho, C., Narayanan, S., and Raghavendra, C. S. *Collaborative Classification Applications in Sensor Networks*. Presentation, August 2002. URL http://sail.usc.edu/pdf/CollaborativeClassificationApplicationsInSensorNetworks_chartchai_SAM2002_08_07_02.pdf. Online. Accessed at 2007-06-23.

[174] Meesookho, C., Narayanan, S., and Raghavendra, C. S. *Collaborative Classification Applications in Sensor Networks*. In: Proceedings of the 2nd IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM), pages 370–374. IEEE Signal Processing Society, Arlington, VA, USA, August 2002.

[175] Michail, A. and Ephremides, A. *Energy-Efficient Routing for Connection-Oriented Traffic in Wireless Ad-Hoc Networks*. In: Mobile Networks and Applications, Vol. 8, No. 5, pages 517–533, October 2003.

[176] Min, R., Bhardwaj, M., Cho, S. H., Ickes, N., Shil, E., Sinha, A., and andf A. Chandrakasan, A. W. *Energy-Centric Enabling Technologies for Wireless Sensor Networks*. In: IEEE Wireless Communications Magazine, Vol. 9, No. 4, pages 28–39, August 2002.

[177] Misra, A. and Banerjee, S. *MRPC: Maximizing Network Lifetime for Reliable Routing in Wireless Environments*. In: Proceedings of the 4th IEEE International Wireless Communications and Networking Conference (WCNC), pages 800–806. IEEE Computer Society, Orlando, FL, USA, March 2002.

[178] Moore, G. *Cramming More Components onto Integrated Circuits*. In: Electronics, Vol. 38, No. 8, pages 114–117, April 1965.

[179] Murthy, S. and Garcia-Luna-Aceves, J. J. *An Efficient Routing Protocol for Wireless Networks*. In: Mobile Networks and Applications, Vol. 1, No. 2, pages 183–197, June 1996.

[180] Nath, S., Gibbons, P. B., Seshan, S., and Anderson, Z. R. *Synopsis Diffusion for Robust Aggregation in Sensor Networks*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 250–262. ACM Press, Baltimore, MD, USA, November 2004.

[181] National Interagency Fire Center. *Historical Wildland Fire Summaries*. URL http://www.nifc.gov/stats/. Online. Accessed at 2007-06-23.

[182] Niculescu, D. and Nath, B. *Ad Hoc Positioning System (APS) using AOA*. In: Proceedings of the 22nd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1734–1743. IEEE Computer Society, San Francisco, CA, UAS, March 2003.

[183] Nikaein, N. and Bonnet, C. *Topology Management for Improving Routing and Network Performances in Mobile Ad Hoc Networks*. In: Mobile Networks and Applications, Vol. 9, No. 6, pages 583–594, December 2004.

[184] Nonnenmacher, J., Biersack, E. W., and Towsley, D. *Parity-Based Loss Recovery for Reliable Multicast Transmission*. In: IEEE/ACM Transactions on Networking, Vol. 6, No. 4, pages 349–361, August 1998.

[185] Park, V. D. and Corson, M. S. *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. In: Proceedings of the 16th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1405–1413. IEEE Computer Society, Kobe, Japan, April 1997.

[186] Patwari, N., Ash, J. N., Kyperountas, S., Hero, A. O., Moses, R. L., and Correal, N. S. *Locating the Nodes: Cooperative Localization in Wireless Sensor Networks*. In: IEEE Signal Processing Magazine, Vol. 22, No. 4, pages 54–69, July 2005.

[187] Pearl, J. *Fusion, Propagation, and Structuring in Belief Networks*. In: Artificial Intelligence, Vol. 29, No. 3, pages 241–288, September 1986.

[188] Peleg, D. and Rubinovich, V. *A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction*. In: SIAM Journal of Computing, Vol. 30, No. 5, pages 1427–1442, May 2000.

[189] Pellegrino, P., Bonino, D., and Corno, F. *Domotic House Gateway*. pages 1915–1920. ACM Press, Dijon, France, April 2006.

[190] Perkins, C. E. and Bhagwat, P. *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. In: Proceedings of the 17th ACM International SIGCOMM, pages 234–244. ACM Press, London, UK, August 1994.

[191] Peterson, L. and Davie, B. S. Computer Networks: A Systems Approach. Morgan Kaufman, 2007.

[192] Peterson, W. W. and Weldon, E. J. Error-Correcting Codes. MIT Press, Cambridge, MA, USA, 1972.

[193] Prim, R. C. *Shortest Connection Networks and Some Generalizations*. In: Bell System Technical Journal, Vol. 36, No. 6, pages 1389–1401, 1957.

[194] Pursley, M. B., Russell, H. B., and Wysocarski, J. S. *Energy-Efficient Routing in Frequency-Hop Radio Networks with Partial-Band Interference*. In: Proceedings of the 2nd IEEE International Wireless Communications and Networking Conference (WCNC), pages 79–83. IEEE Computer Society, Chigago, IL, USA, September 2000.

[195] Qureshi, F. and Terzopoulos, D. *Virtual Vision and Smart Camera Networks*. In: Proceedings of the International Workshop on Distributed Smart Cameras (DSC), pages 62–66. Boulder, CO, USA, October 2006.

[196] Raghunathan, V., Schurgers, C., Park, S., and Srivastava, M. B. *Energy-Aware Wireless Microsensor Networks*. In: IEEE Signal Processing Magazine, Vol. 19, No. 2, pages 40–50, March 2002.

[197] Ramanathan, R. and Rosales-Hain, R. *Topology Control of Multihop Wireless Networks using Transmit Power Adjustment*. In: Proceedings of the 19th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 404–413. IEEE Computer Society, Tel-Aviv, Israel, March 2000.

[198] Rappaport, T. Wireless Communications: Principles and Practice. Prentice Hall, 2001.

[199] Reed, I. and Solomon, G. *Polynomial Codes Over Certain Finite Fields*. In: SIAM Journal on Applied Mathematics, Vol. 8, No. 2, pages 300–304, June 1960.

[200] RF Monolithics, Inc. 868.35 MHz Hybrid Transceiver TR1001. URL http://www.rfm.com/products/data/tr1001.pdf. Online. Accessed at 2007-06-23.

[201] RF Monolithics, Inc. ASH Transceiver Designer's Guide. URL http://www.rfm.com/products/tr_des24.pdf. Online. Accessed at 2007-06-23.

[202] Richardson, T., Shokrollahi, M. A., and Urbanke, R. *Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes*. In: IEEE Transactions on Information Theory, Vol. 47, No. 2, pages 619–637, February 2001.

[203] Richardson, T. and Urbanke, R. *Efficient Encoding of Low-Density Parity-Check Codes*. In: IEEE Transactions on Information Theory, Vol. 47, No. 2, pages 638–656, September 2001.

[204] Rodoplu, V. and Meng, T. H. *Minimum Energy Mobile Wireless Networks*. In: IEEE Journal on Selected Areas in Communications, Vol. 17, No. 8, pages 1333–1344, August 1999.

[205] Ruzzelli, A. G., O'Hare, G. M. P., O'Grady, M. J., and Tynan, R. *MERLIN: A Synergetic Integration of MAC and Routing Protocol for Distributed Sensor Networks*. In: Proceedings of the 3rd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), pages 266–275. IEEE Computer Society, Reston, VA, USA, September 2006.

[206] Sadler, C. *ZebraNet*. URL http://cmsadler.googlepages.com/. Online. Accessed at 2007-06-23.

[207] Sallai, J., Balogh, G., Maróti, M., Lédeczi, A., and Kusy, B. *Acoustic Ranging in Resource-Constrained Sensor Networks*. In: Proceedings of the International Conference on Wireless Networks (ICWN), pages 467–472. Las Vegas, NV, USA, June 2004.

[208] Sankar, A. and Liu, Z. *Maximum Lifetime Routing in Wireless Ad-Hoc Networks*. In: Proceedings of the 23rd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1089–1097. IEEE Computer Society, Hong Kong, China, March 2004.

[209] Saukh, O., Marrón, P. J., Lachenmann, A., Gauger, M., Minder, D., and Rothermel, K. *Generic Routing Metric and Policies for WSNs*. In: Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN), pages 99–114. Springer, Zuerich, Switzerland, February 2006.

[210] ScatterWeb. The Modular Sensor Board. URL http://www.scatterweb.de/content/downloads/datasheets/fact-sheet-msb430-v1.0-en.pdf. Online. Accessed at 2007-06-23.

[211] Schiller, J., Liers, A., Ritter, H., Winter, R., and Voigt, T. *ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing*. In: Proceedings of the 38th IEEE Hawaii International Conference on System Sciences (HICSS), pages 286–294. IEEE Computer Society, Maui, HI, USA, January 2005.

[212] Schurgers, C. and Srivastava, M. B. *Energy-Efficient Routing in Wireless Sensor Networks*. In: Proceedings of the IEEE Military Communications Conference (MILCOM), pages 357–361. IEEE Computer Society, Vienne, VA, USA, October 2001.

[213] Schurgers, C., Tsiatsis, V., Ganeriwal, S., and Srivastava, M. *Optimizing Sensor Networks in the Energy-Latency-Density Design Space*. In: IEEE Transactions on Mobile Computing, Vol. 1, No. 1, pages 70–80, January 2002.

[214] Schurgers, C., Tsiatsis, V., Ganeriwal, S., and Srivastava, M. *Topology Management for Sensor Networks: Exploiting Latency and Density*. pages 135–145. ACM Press, Lausanne, Switzerland, June 2002.

[215] Schwartz, J. W. and Wolf, J. K. *A Systematic (12,8) Code for Correcting Single Errors and Detecting Adjacent Errors*. In: IEEE Transactions on Computers, Vol. 39, No. 11, pages 1403–1404, November 1990.

[216] Schwiebert, L., Gupta, S. K. S., and Weinmann, J. *Research Challenges in Wireless Networks of Biomedical Sensors*. In: Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 151–165. ACM Press, Rome, Italy, July 2001.

[217] Seada, K., Zuniga, M., Helmy, A., and Krishnamachari, B. *Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 108–121. ACM Press, Baltimore, MD, USA, November 2004.

[218] Shah, R. and Rabaey, J. *Energy-Aware Routing for Low-Energy Ad Hoc Sensor Networks*. In: Proceedings of the 4th IEEE International Wireless Communications and Networking Conference (WCNC), pages 350–355. IEEE Computer Society, Orlando, FL, USA, March 2002.

[219] Shannon, C. E. *A Mathematical Theory of Communication*. In: Bell System Technical Journal, Vol. 27, pages 379–423 and 623–656, July and October 1948.

[220] Sheth, A., Thekkath, C. A., Mehta, P., Tejaswi, K., Parekh, C., Singh, T. N., and Desai, U. B. *SenSlide: A Distributed Landslide Prediction System*. In: ACM SIGOPS Operating Systems Review, Vol. 41, No. 2, pages 75–87, April 2007.

[221] Shokrollahi, A. *Raptor Codes*. In: IEEE/ACM Transactions on Networking, Vol. 14, No. SI, pages 2551–2567, June 2006.

[222] Shrivastava, N., Buragohain, C., and Agrawal, D. *Medians and Beyond: New Aggregation Techniques for Sensor Networks*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 239–249. ACM Press, Baltimore, MD, USA, November 2004.

[223] Shukla, S., Bulusu, N., and Jha, S. *Cane-Toad Monitoring in Kakadu National Park Using Wireless Sensor Networks*. In: Proceedings of the 18th Asia-Pacific Advanced Network Meeting (APAN). Cairns, Australia, July 2004.

[224] Simon, G., Maróti, M., Lédeczi, A., Balogh, G., Kusy, B., Nádas, A., Pap, G., Sallai, J., and Frampton, K. *Sensor Network-Based Countersniper System*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 1–12. ACM Press, Baltimore, MD, USA, November 2004.

[225] Singh, S., Woo, M., and Raghavendra, C. S. *Power-Aware Routing in Mobile Ad Hoc Networks*. In: Proceedings of the 4th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 181–190. ACM Press, Dallas, TX, USA, October 1998.

[226] Sproull, R. F. and Cohen, D. *High-Level Protocols*. In: Proceedings of the IEEE, Vol. 66, No. 11, pages 1371–1386, November 1978.

[227] Srinivasan, V., Chiasserini, C.-F., Nuggehalli, P. S., and Rao, R. R. *Optimal Rate Allocation for Energy-Efficient Multipath Routing in Wireless Ad Hoc Networks*. In: IEEE Transactions on Wireless Communications, Vol. 3, No. 3, pages 891–899, May 2004.

[228] Stann, F., Heidemann, J., Shroff, R., and Murtaza, M. Z. *RBP: Robust Broadcast Propagation in Wireless Networks*. In: Proceedings of the 4th ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 85–98. ACM Press, Boulder, CO, USA, October 2006.

[229] Stojmenovic, I. and Lin, X. *Power-Aware Localized Routing in Wireless Networks*. In: IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 11, pages 1122–1133, November 2001.

[230] Stoleru, R., He, T., Stankovic, J. A., and Luebke, D. P. *A High-Accuracy, Low-Cost Localization System for Wireless Sensor Networks*. In: Proceedings of the 3rd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 13–26. ACM Press, San Diego, CA, USA, November 2005.

[231] Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., and Culler, D. *An Analysis of a Large Scale Habitat Monitoring Application*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 214–226. ACM Press, Baltimore, MD, USA, November 2004.

[232] Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A., and Estrin, D. *Habitat Monitoring with Sensor Networks*. In: Communications of the ACM, Vol. 47, No. 6, pages 34–40, June 2004.

[233] Tabar, A. M., Keshavarz, A., and Aghajan, H. *Smart Home Care Network Using Sensor Fusion and Distributed Vision-Based Reasoning*. In: Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks (VSSN), pages 145–154. ACM Press, Santa Barbara, CA, USA, October 2006.

[234] Takahasi, H. and Matsuyama, A. *An Approximation Solution for the Steiner Tree Problemin Graphs*. In: Mathematica Japonica, Vol. 24, No. 6, pages 573–577, 1980.

[235] Tavli, B. and Heinzelman, W. B. *Energy and Spatial Reuse Efficient Network-Wide Real-Time Data Broadcasting in Mobile Ad Hoc Networks*. In: IEEE Transactions on Mobile Computing, Vol. 5, No. 10, pages 1297–1312, October 2006.

[236] The Defense Advanced Research Projects Agency. *Technical Offices Programs*. URL http://www.darpa.mil/body/off_programs.html. Online. Accessed at 2007-06-23.

[237] Thorstensen, B., Syversen, T., Bjørnvold, T.-A., and Walseth, T. *Electronic Shepherd - A Low-Cost, Low-Bandwidth, Wireless Network System*. In: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications, and Services (MOBISYS), pages 245–255. ACM Press, Boston, MA, USA, June 2004.

[238] Toh, C.-K. *Maximum Battery Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks*. In: IEEE Communications Magazine, Vol. 39, No. 6, pages 138–147, June 2001.

[239] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., and Hong, W. *A Macroscope in the Redwoods*. In: Proceedings of the 3rd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 51–63. ACM Press, San Diego, CA, USA, November 2005.

[240] Upadhyayula, S., Annamalai, V., and Gupta, S. K. S. *A Low-Latency and Energy-Efficient Algorithm for Convergecast in Wireless Sensor Networks*. In: Proceedings of the 46th IEEE Global Telecommunications Conference (GLOBECOM), pages 3525–3530. IEEE Communication Society, San Francisco, CA, USA, December 2003.

[241] van Dam, T. and Langendoen, K. *An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks*. In: Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 171–180. ACM Press, Los Angeles, CA, USA, November 2003.

[242] Wang, H.-L., Miao, J., and Chang, M. *An Enhanced IEEE 802.11 Retransmission Scheme*. In: Proceedings of the 5th IEEE International Wireless Communications and Networking Conference (WCNC), pages 66–71. IEEE Computer Society, New Orleans, LA, USA, March 2003.

[243] Wang, L. and Kulkarni, S. S. *Proactive Reliable Bulk Data Dissemination in Sensor Networks*. In: Proceedings of the 17th International Conference on Parallel and Distributed Computing and Systems (PDCS), pages 773–778. ACTA Press, Phoenix, AZ, USA, November 2005.

[244] Wang, Y., Martonosi, M., and Peh, L.-S. *Supervised Learning in Sensor Networks: New Approaches with Routing, Reliability Optimizations*. In: Proceedings of the 3rd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), pages 256–265. IEEE Computer Society, Reston, VA, USA, September 2006.

[245] Warneke, B., Last, M., Liebowitz, B., and Pister, K. S. J. *Smart Dust: Communicating with a Cubic-Millimeter Computer*. In: IEEE Computer, Vol. 31, No. 1, pages 44–51, January 2001.

[246] Wattenhofer, R., Li, L., Bahl, P., and Wang, Y.-M. *Distributed Topology Control for Power-Efficient Operations in Multihop Wireless Ad Hoc Networks*. In: Proceedings of the 20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1388–1397. IEEE Computer Society, Anchorage, AK, USA, April 2001.

[247] Welsh, M. *Deploying a Sensor Network on an Active Volcano*. Presentation, 2006. URL http://iic.harvard.edu/downloads/seminar_welsh_100406.pdf. Online. Accessed at 2007-06-23.

[248] Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., and Welsh, M. *Monitoring Volcanic Eruptions with a Wireless Sensor Network*. In: Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN), pages 108–120. IEEE Press, Istanbul, Turkey, 2005.

[249] Werner-Allen, G., Lorincz, K., Welsh, M., Marcillo, O., Johnson, J., Ruiz, M., and Lees, J. *Deploying a Wireless Sensor Network on an Active Volcano*. In: IEEE Internet Computing, Vol. 10, No. 2, pages 18–25, March/April 2006.

[250] Wesson, R. B., Hayes-Roth, F. A., Burge, J. W., Stasz, C., and Sunshine, C. A. *Network Structures for Distributed Situation Assessment*. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, pages 5–23, January 1981.

[251] Westhoff, D., Girao, J., and Acharya, M. *Concealed Data Aggregation for Reverse Multicast Traffic in Sensor Networks: Encryption, Key Distribution, and Routing Adaptation*. In: IEEE Transactions on Mobile Computing, Vol. 5, No. 10, pages 1417–1431, October 2006.

[252] Wikipedia. The Free Encyclopedia. *ALERT - Automated Local Evaluation in Real-Time*. URL http://www.alertsystems.org/. Online. Accessed at 2007-06-23.

[253] Wikipedia. The Free Encyclopedia. *Ecophysiology*. URL http://en.wikipedia.org/wiki/Ecophysiology. Online. Accessed at 2007-06-23.

[254] Wikipedia. The Free Encyclopedia. *The Leach's Storm-Petrel*. URL http://en.wikipedia.org/wiki/Leach%27s_Storm-Petrel. Online. Accessed at 2007-06-23.

[255] Wikipedia. The Free Encyclopedia. *Shannon-Hartley Theorem*. URL http://en.wikipedia.org/wiki/Shannon-Hartley_theorem. Online. Accessed at 2007-06-23.

[256] Wikipedia. The Free Encyclopedia. *Universal Asynchronous Receiver/Transmitter*. URL http://en.wikipedia.org/wiki/UART. Online. Accessed at 2007-06-23.

[257] Willig, A. and Mitschke, R. *Results of Bit Error Measurements with Sensor Nodes and Casuistic Consequences for Design of Energy-Efficient Error Control Schemes*. In: Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN), pages 310–325. Springer, Zurich, Switzerland, February 2006.

[258] Woo, A., Tong, T., and Culler, D. *Taming the Underlying Challenges for Reliable Multihop Routing in Sensor Networks*. In: Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 14–27. ACM Press, Los Angeles, CA, USA, November 2003.

[259] Xu, N., Rangwala, S., Chintalapudi, K., Ganesan, D., Broad, A., Govindan, R., and Estrin, D. *A Wireless Sensor Network for Structural Monitoring*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 13–24. ACM Press, Baltimore, MD, USA, November 2004.

[260] Xu, Y., Bien, S., Mori, Y., Heidemann, J., and Estrin, D. *Topology Control Protocols to Conserve Energy in Wireless Ad Hoc Networks*. Technical Report 6, Center for Embedded Networked Sensing, UCLA, January 2003.

[261] Xu, Y., Heidemann, J., and Estrin, D. *Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks*. Technical Report 527, Information Sciences Institute, USC, October 2000.

[262] Xu, Y., Heidemann, J., and Estrin, D. *Geography-Informed Energy Conservation for Ad-Hoc Routing*. In: Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 70–84. ACM Press, Rome, Italy, July 2001.

[263] Xue, Y., Cui, Y., and Nahrstedt, K. *Maximizing Lifetime for Data Aggregation in Wireless Sensor Networks*. In: Mobile Networks and Applications, Vol. 10, No. 6, pages 853–864, December 2005.

[264] Yang, Y., Wang, X., Zhu, S., and Cao, G. *SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks*. In: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), pages 356–367. ACM Press, Florence, Italy, May 2006.

[265] Yarvis, M., Conner, W., Krishnamurthy, L., Chhabra, J., Elliott, B., and Mainwaring, A. *Real-World Experiences with an Interactive Ad Hoc Sensor Network*. In: Proceedings of the 31st IEEE International Conference on Parallel Processing Workshops (ICPPW), pages 143–152. IEEE Computer Society, Vancouver, BC, Canada, August 2002.

[266] Ye, F., Chen, A., Liu, S., and Zhang, L. *A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks*. In: Proceedings of the 12th IEEE International Conference on Computer Communications and Networks (ICCCN), pages 304–309. IEEE Computer Society, Scottsdale, AZ, USA, October 2001.

[267] Ye, F., Luo, H., Cheng, J., Lu, S., and Zhang, L. *A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks*. In: Proceedings of the 8th ACM International Conference on Mobile Computing and Networking (MOBICOM), pages 148–159. ACM Press, Atlanta, GA, USA, September 2002.

[268] Ye, F., Yi, S., and Sikdar, B. *Improving Spatial Reuse of IEEE 802.11-Based Ad Hoc Networks*. In: Proceedings of the 46th IEEE Global Telecommunications Conference (GLOBECOM), pages 1013–1017. IEEE Computer Society, San Francisco, CA, USA, December 2003.

[269] Ye, F., Zhong, G., Lu, S., and Zhang, L. *PEAS: A Robust Energy Conserving Protocol for Long-Lived Sensor Networks*. In: Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS), pages 200–210. IEEE Press, Providence, RI, USA, May 2003.

[270] Ye, W., Heidemann, J., and Estrin, D. *An Energy-Efficient MAC Protocol for Wireless Sensor Networks*. In: Proceedings of the 21st Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 1567–1576. IEEE Computer Society, New York, NY, USA, June 2002.

[271] Ye, W., Silva, F., and Heidemann, J. *Ultra-Low Duty Cycle MAC with Scheduled Channel Polling*. In: Proceedings of the 4th ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 321–334. ACM Press, Boulder, CO, USA, October 2006.

[272] Younis, O. and Fahmy, S. *HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad-Hoc Sensor Networks*. In: IEEE Transactions on Mobile Computing, Vol. 3, No. 4, pages 366–379, October 2004.

[273] Zhang, H. and Hou, J. C. *Maximizing α-Lifetime for Wireless Sensor Networks*. In: Proceedings of the 3rd International Workshop on Measurement, Modeling, and Performance Analysis of Wireless Sensor Networks (SENMETRICS), pages 70–77. San Diego, CA, USA, July 2005.

[274] Zhang, P., Sadler, C. M., Lyon, S. A., and Martonosi, M. *Hardware Design Experiences in ZebraNet*. In: Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 227–238. ACM Press, Baltimore, MD, USA, November 2004.

[275] Zhao, B. and Valenti, M. C. *Practical Relay Networks: A Generalization of Hybrid-ARQ*. In: IEEE Journal on Selected Areas in Communications, Vol. 23, No. 1, pages 7–18, January 2005.

[276] Zhao, J. and Govindan, R. *Understanding Packet Delivery Performance in Dense Wireless Sensor Networks*. In: Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems (SENSYS), pages 1–13. ACM Press, Los Angeles, CA, USA, November 2003.

[277] Zhong, Z. and Nelakuditi, S. *On the Efficacy of Opportunistic Routing*. In: Proceedings of the 4th IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), pages 441–450. IEEE Computer Society, San Diego, CA, USA, June 2007.

[278] Zhou, G., He, T., Krishnamurthy, S., and Stankovic, J. A. *Impact of Radio Irregularity on Wireless Sensor Networks*. In: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications, and Services (MOBISYS), pages 125–138. ACM Press, Boston, MA, USA, June 2004.

[279] Zorzi, M. and Armaroli, A. *Advancement Optimization in Multihop Wireless Networks*. In: Proceedings of the 58th IEEE International Vehicular Technology Conference (VTC), pages 2891–2894. IEEE Press, Orlando, FL, USA, October 2003.

[280] Zorzi, M. and Rao, R. R. *Capture and Retransmission Control in Mobile Radio*. In: IEEE Journal on Selected Areas in Communications, Vol. 12, No. 8, pages 1289–1298, October 1994.

[281] Zorzi, M. and Rao, R. R. *Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Energy and Latency Performance*. In: IEEE Transactions on Mobile Computing, Vol. 2, No. 4, pages 349–365, October 2003.

[282] Zuniga, M. and Krishnamachari, B. *Analyzing the Transitional Region in Low Power Wireless Links*. In: Proceedings of the 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON), pages 517–526. IEEE Computer Society, Santa Clara, CA, USA, October 2004.