Masterarbeit

An Energy Harvesting Networking Service for Wireless Sensor Networks using Network Coding

Stefan Ruff, BSc

Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Begutachter: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß
Betreuer: Dipl.-Ing. Philipp Maria Glatz, Bakk.techn.

Graz, im Oktober 2010

Kurzfassung

Obwohl in den letzten Jahren einiges an Forschung im Bereich von Wireless Sensor Networks (WSNs) betrieben wurde, ist die begrenzte Stromversorgung von batteriebetriebenen WSNs noch immer ein großes Problem. Energy Harvesting Vorrichtungen bieten eine Alternative zur begrenzten Stromversorgung durch Batterien. Während es durch Energy Harvesting möglich ist, die Probleme von verbrauchten Batterien und dem Ersetzen dieser zu eliminieren, gibt es andere Probleme und Limitierungen die nun beachtet werden müssen.

Im optimalen Fall sollten Energy Harvesting WSNs in der Lage sein, ihren Betrieb unendlich lang fortzusetzen solange keine technischen Fehler eintreten. Diese Diplomarbeit präsentiert das Design und die Implementierung eines Networking Frameworks für Energy Harvesting WSNs. Das Hauptziel des Networking Frameworks ist es, alle Knoten eines Netzwerk energieneutral zu betreiben. Um dieses Zeit zu erreichen implementiert das präsentierte Networking Framework mehrere Algorithmen die sich in die Kategorien Networking und reduzieren des Stromverbrauchs aufteilen lassen.

In der Networking Schicht implementiert das Framework Energy Harvesting Aware Routing, Network Coding und Opportunistic Network Coding. Energy Harvesting Aware Routing ist ein probabilistisches Routingverfahren, das versucht, den anfallenden Netzwerktraffic entsprechend der Energieprofile der Knoten des Netzwerks zu verteilen. Network Coding arbeitet anstelle von Energy Harvesting Aware Routing und reduziert den anfallenden Netzwerktraffic durch das Kombinieren von Paketen. Opportunistic Network Coding versucht in den Datenflüssen von Energy Harvesting Aware Routing Möglichkeiten zum Kombinieren von Paketen zu detektieren. Um den Stromverbrauch zu reduzieren wird Duty Cycling auf den Knoten des Netzwerks implementiert und zusätzlich verwendet das Networking Framework die bereits existierende Implementierung von LPL um den Stromverbrauch im Leerlauf zu reduzieren.

Diese Diplomarbeit präsentiert einen Einblick in bereits existierende Arbeit zu verschiedenen Themen im Bereich von WSNs mit einem speziellen Fokus auf Energy Harvesting und zeigt den Bedarf für ein Energy Harvesting Networking Framework für WSNs. Die Ergebnisse zeigen, dass das vorgeschlagene Framework eine gute Lösung für die Probleme von Energy Harvesting WSNs bietet. Speziell Opportunistic Network Coding zeigt sehr großes Potential da es ermöglicht, die Anzahl der notwendigen Übertragungen zu reduzieren ohne die Flexibilität des darunter liegenden Routingverfahrens zu verlieren.

Abstract

While a lot of research has taken place on the topic of wireless sensor networks (WSNs) in the last years, the limited power supply of battery powered WSNs is still a major problem. Energy harvesting devices as a power supply for the sensor nodes offer an alternative to the limited battery power supply. While energy harvesting devices remove the problem of depleted power supplies and battery replacement, there are other problems and limitations that have to be considered.

In an optimal case, deployed energy harvesting WSNs should be able to run infinitely as long as there are no technical faults. This master thesis presents the design and implementation of a networking framework for energy harvesting WSNs. The main goal of the networking framework is to operate all nodes in the network in energy neutral mode to avoid nodes running out of power. To achieve this goal, the networking framework implements several algorithms that can be divided into two main categories, networking and power saving.

On the networking layer, the framework implements energy harvesting aware routing, network coding and opportunistic network coding. Energy harvesting aware routing is a probabilistic routing scheme that aims to balance network traffic according to the energy profiles of the nodes. While network coding operates alongside energy harvesting aware routing to reduce the needed packet transmissions opportunistic network coding is detecting possibilities to combine packets in the data flows of the energy harvesting aware routing algorithm. For power saving the framework implements duty cycling of the sensor nodes and uses an existing low power listening implementation that both help to reduce the power that the nodes waste in idle mode.

This master thesis presents an insight on related work on several topics in the field of WSNs with a special focus on energy harvesting awareness and shows the need for an energy harvesting networking framework for WSNs. The results show that the proposed framework is a viable solution for the problem of energy harvesting powered WSNs. Special opportunistic network coding shows a very great potential, since it allows reducing the amount of packet transmissions while still keeping the flexibility of the underlying routing algorithm.

Danksagung

Diese Diplomarbeit wurde im Studienjahr 2009/10 am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Ich möchte mich vor allem bei Dipl.-Ing. Philipp Glatz, Bakk.techn. für die sehr engagierte und freundliche Betreuung bedanken, durch die diese Arbeit erst möglich war. Des weiteren gilt mein Dank auch Herrn O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß für die Begutachtung meiner Diplomarbeit. Ich möchte mich aber auch bei allen Kollegen am Institut für Technische Informatik für die angenehme Zeit, die ich während meiner Diplomarbeit hier hatte, bedanken.

Ganz besonders möchte ich mich auch bei meinen Eltern bedanken, ohne deren Unterstützung mein Studium nicht möglich gewesen wäre und die mir immer zur Seite standen wann immer ich ihre Hilfe benötigte.

Contents

1 Introduction **9**
1.1 Motivation . . . 9
1.2 Goal . . . 9
1.3 Structure . . . 10

2 Related Work **11**
2.1 Routing . . . 11
2.1.1 Routing Design Criteria . . . 11
2.1.2 Power Aware Routing . . . 14
2.1.3 Energy Harvesting Aware Routing . . . 18
2.2 Network Coding . . . 23
2.2.1 Basics . . . 23
2.2.2 Network Coding in WSNs . . . 25
2.2.3 Opportunistic Network Coding . . . 26
2.3 Node Power Saving Strategies . . . 27
2.3.1 Duty Cycling . . . 27
2.3.2 Low Power Listening . . . 32
2.3.3 Balancing Residual Energy Amongst Sensor Nodes . . . 35
2.4 Existing Middleware Implementations . . . 36
2.4.1 Maté . . . 37
2.4.2 Impala . . . 38
2.4.3 SINA . . . 38
2.4.4 MiLAN . . . 39
2.4.5 Mires . . . 39
2.5 Summary . . . 40

3 Design **41**
3.1 Basic Design . . . 41
3.1.1 Problems . . . 42
3.1.2 Goals . . . 44
3.1.3 Parts of the Networking Framework . . . 45
3.2 Energy Harvesting Aware Routing . . . 47
3.3 Network Coding . . . 51
3.3.1 Opportunistic Network Coding . . . 53
3.4 Duty Cycling . . . 56

3.5 Low Power Listening 58
3.6 Testing 58

4 Implementation **60**
4.1 Implementation Scheme 60
4.1.1 Structure 60
4.1.2 Interfaces 62
4.1.3 Test Scenarios 62
4.1.4 Installation 63
4.2 Energy Harvesting Aware Routing 63
4.3 Network Coding 67
4.4 Opportunistic Network Coding 69
4.5 Duty Cycling 75
4.5.1 Low Power Listening 80

5 Results **81**
5.1 Comparison of the three Networking Types 81
5.1.1 Comparison without Duty Cycling 85
5.1.2 Measurements of data sending on intermediate node 86
5.1.3 Influence of the data sending rate 89
5.1.4 Performance under different reception strengths 89
5.1.5 Changing the networking type 90
5.1.6 Influence of the sending order on the average delay 91
5.2 Energy Harvesting Aware Routing 92
5.2.1 Energy Level Adaption 92
5.3 Opportunistic Network Coding 93
5.3.1 Influence of the Maximum Wait Time 93
5.4 Comparison of EHAR and ONC in a single data sink network 95
5.5 Duty Cycling and Low Power Listening 97

6 Conclusion **101**
6.1 Summary 101
6.2 Future Work 102

A List of Abbrevations **103**

Bibliography **104**

List of Figures

2.1 Simple network setup without and with NC 25

3.1 Network configuration for $n \times n$ nodes 42
3.2 Layer model of the networking framework 46
3.3 Example setup for EHAR 50
3.4 Example setup for NC 53
3.5 Possible cases for using ONC with two packets 55

4.1 Interactions between the modules of the framework 61
4.2 Flow diagram of the EHAR module 64
4.3 Flow diagram of the NC module 68
4.4 Flow diagram of the ONC module 69
4.5 Flow diagram of the duty cycling module 76

5.1 Total amount of packets sent, received and overheard per node with NC with activated duty cycling 81
5.2 Total amount of packets sent, received and overheard per node with EHAR with activated duty cycling 82
5.3 Total amount of packets sent, received and overheard per node with ONC with activated duty cycling 82
5.4 Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling 83
5.5 Comparison of the sent, received and overheard packets per node in a 6×6 network with activated duty cycling for NC, EHAR and ONC 84
5.6 Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling for a network with 6×6 nodes 85
5.7 Comparison of the sent, received and overheard packets per node in a 4×4 network without duty cycling for NC, EHAR and ONC 86
5.8 Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type without duty cycling 87
5.9 Data flow on intermediate node with NC 87
5.10 Data flow on intermediate node with EHAR 88
5.11 Data flow on intermediate node with ONC 88

5.12 Influence of the sending rate of the user application on the amount of received packets and the total average delay for NC, EHAR and ONC in a 4×4 network with activated duty cycling 89
5.13 Influence of the reception strength in the network on the amount of received packets on their destination for the different networking types with activated duty cycling 90
5.14 Timeline of sent packets and total amount of sent packets per type per node for changing the networking type from EHAR to NC in a 4×4 network with activated duty cycling 91
5.15 Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with inverted sending order and activated duty cycling 91
5.16 Comparison of the amount of sent, received and overheard packet for a network with the same energy level on all nodes and a network with one node with a lower energy value 92
5.17 Average delay between enqueuing a packet at the source and receiving the packet at the destination depending on the maximum wait time for ONC before transmitting a packet uncombined with activated duty cycling 93
5.18 Average delay between enqueuing a packet at the source and receiving the packet at the destination depending on the maximum wait time for ONC before transmitting a packet uncombined without duty cycling 93
5.19 Amount of packets that were sent out combined by opportunistic network coding depending on the maximum wait time for ONC with activated duty cycling 94
5.20 Amount of packets that were sent out combined by opportunistic network coding depending on the maximum wait time for ONC without duty cycling 94
5.21 Influence of the maximum wait time on the total amount of combined packets and average delay for ONC in a 5×5 network with activated duty cycling 95
5.22 Total amount of packets sent, received and overheard per node with ONC with activated duty cycling in a network with a single data sink 96
5.23 Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling in a network with a single data sink 97
5.24 Measurement of one listening process with LPL 98
5.25 Comparison of one LPL listening interval to the same time listening without LPL 99
5.26 Initial costs for activating LPL and switching to active in the DC module . 100

Chapter 1

Introduction

1.1 Motivation

With the advances in research on topics like microprocessors, miniaturization, reduced power consumption, wireless communication and the reduced production costs of these parts, Wireless Sensor Networks (WSNs) were not only made possible but are an increasingly popular topic of both research and real-world applications. While it is possible to deploy WSNs independent of any connection to a constant power supply or a fixed network, there are some constraints for WSNs. To be able to operate without any connection to an extern power supply, WSNs have to rely on their battery driven power supply. Even if the advances in reduced power consumption and increased battery size can increase the lifetime of a WSN, a battery driven power supply still remains limited and will run out of power sooner or later. But especially when WSNs are deployed in hard to reach regions, simply just replacing depleted batteries is not viable. Another constraint for WSNs is the limited available network bandwidth because radio communication is energy intensive and interferences and congestion further limit the available bandwidth.

One solution to the power supply problem of WSNs is the use of energy harvesting as a power supply. Energy harvesting devices are able to supply the connected WSNs with a continuous income of power that will not get depleted unlike batteries. But the amount of power that can be provided by an energy harvesting device is very limited and will not be able to sustain all workloads that were possible in battery powered WSNs. But if the WSN is able to reduce its power consumption to stay within the limits of the power provided by the energy harvesting device, WSNs are able to run indefinitely.

1.2 Goal

While in theory an energy harvesting powered WSN is able to operate infinitely as long as there are no technical defects on the hardware, this does not hold true in reality. The power that is provided by an energy harvesting device is very limited and can vary strongly over time. The goal of this thesis is to analyze the existing technologies on various topics of research for WSNs and propose a networking framework that is specifically designed for the needs of energy harvesting powered WSNs. The focus of the networking framework will be on two main areas of research. The first area is a networking implementation that is specifically designed to the needs of energy harvesting and able to balance the network traffic and to reduce the amount of required packet transmissions. The second area is the power consumption of the nodes themselves. The networking framework provides possibilities to reduce the power consumption in the network to allow the nodes to operate energy neutral.

1.3 Structure

The remaining chapters of this thesis are organized as follows. Chapter 2 focuses on related work on different topics in the field of networking and wireless sensor networks (WSNs) that are of importance for the proposed networking framework. The topics that are presented in the chapter are power and energy harvesting aware routing (EHAR), network coding (NC) and opportunistic network coding (ONC), duty cycling, low power listening (LPL) and middleware for WSNs. Chapter 3 presents the design of the networking framework that is proposed in this thesis and Chapter 4 then focuses on the implementation of the proposed networking framework. The evaluation of the performance of the networking framework and the results of the performed tests are presented in Chapter 5. Finally Chapter 6 contains a short summary on the presented work and an outlook on future work on the topic.

Chapter 2

Related Work

WSNs usually consist of low-power sensor nodes with rather small applications running on them that monitor some real world data. Since the power supply of the sensor nodes is limited and a long lifespan of a deployed sensor network is wanted, the power consumption of the sensor nodes should be low to increase the lifetime of the network. This chapter focuses on related work on different aspects of WSNs. It can be categorized into four major topics. The first section focuses on routing for WSNs. The second section deals with NC. The third section presents ways to reduce the power needed on a node and the last section focuses on existing middleware implementations for WSNs.

2.1 Routing

This section focuses on the design of routing algorithms for WSNs. Firstly, this section presents criteria for the design of routing algorithms for WSNs. Secondly, different power aware routing strategies are discussed and some power aware routing algorithms are presented in more detail. Finally, this section focuses on the concept of EHAR and presents already existing EHAR algorithms.

2.1.1 Routing Design Criteria

Routing protocols that are well suited for wireless networks like 802.11 are not necessarily suited for WSNs as well since the design criteria for the routing protocols are different. According to [3][59] the design criteria for WSNs can be separated into seven main categories. The following subsections present those categories and explain why they are important for WSNs and where the criteria differ from other wireless networks.

Routing Targets

Unlike in 802.11 networks there is often no need for all-to-all communication in WSNs, a low rate data monitoring application does not need to be able to address all nodes in the network directly. Furthermore, since the storage space on sensor nodes is limited it is not always possible to store routing information for every sensor node, especially in large WSNs. Rather there usually exist one or more base stations in a WSN that collect the measured data from the sensor nodes in the network. When there exist multiple base stations in a sensor network, there are two possible ways to deal with this. It is either possible that all nodes in the network store routing information for all base stations that exist in the network or that each node in the network only stores routing information for the base station that it is closest to. The second case implies that it is not of importance towards which base station the data of a node is transmitted to.

Single Hop versus Multi Hop Routing

While single hop routing is the standard routing type in most 802.11 networks, for WSNs multi hop routing is the dominant routing form. While single hop routing is favorable when looking at the complexity, there are two main reasons for the popularity of multi hop routing in WSNs. The first reason is that in sensor networks, where the different sensor nodes are distributed over a large area, it is often not possible that all nodes can reach the desired receiver within one hop. The second reason is the power loss of the radio channel that is growing with the distance between sender and receiver. According to [56] the received power P_r at a node can be approximated by:

$$P_r = P_0 \left(\frac{d}{d_0} \right)^{-n} \tag{2.1}$$

In this equation, P_0 is the power received at a small distance d_0 from the transmitter and d is the distance between the transmitter and the receiver. The constant n is the path loss exponent of the wireless channel and usually ranges between 2 and 4 [57]. This makes multi hop routing more energy effective then single hop routing. Adding an additional hop between two nodes is useful as long as the sum of energy costs on all nodes along the multi-hop path stays below the energy cost of the single hop transmission. Limits to the amount of hops between two nodes are set by the network topology and the minimum transmission power of a node. If a node is already transmitting with its minimum transmission power and can still reach the desired receiver then any additional hop between those two nodes would just increase the needed transmission power.

Scalability

Scalability is a very important criteria for WSNs. The amount of sensor nodes in a network can range from just a few nodes to several hundred or more, most sensor networks so far are still in the range below one hundred nodes though. While it is important that a routing protocol for WSNs is able to scale good with the amount of sensor nodes in the network, a routing protocol that performs well in small networks might not perform that well in large networks and vice versa. The density of the network can also be a problem for routing algorithms; a very dense network can require a different routing algorithm then a sparse network.

Mobility

Many routing algorithms assume that the base station and all the nodes in a network are stationary all the time after the start-up. While this simplifies the design for a routing algorithm, it renders the routing algorithm unviable for many real world deployments of sensor networks. Depending on the application, both the sensor nodes [26] [6] [49] and the base station [12] [44] [45] can be mobile in WSNs. Also, the amount of mobile nodes in the network and the frequency of the location changes are of importance for the design of a routing algorithm.

Fault Tolerance

There are many sources of errors in WSNs. A drained power supply, physical damage, environmental interferences or disturbances on the radio channel may cause a sensor node to be temporarily or permanently unavailable or completely out of order. Since a routing protocol cannot prevent hardware defects on a sensor node the routing algorithm needs to be able to react to failures in the network and recover from them. Because as long as there are enough nodes alive to enable a requested connection, the routing protocol should be able to keep the routing alive. In [52] an overview on possible ways for fault management in WSNs is presented. One possibility to deal with failing nodes in a network is to reroute the traffic over a different sensor node that is still alive [48]. Another possibility to reduce the impact of failing nodes on the network is to have multiple redundant links towards a data sink like it is the case with multipath routing [51].

Flat-based versus Hierarchical-based Routing

The organization of the sensor nodes in a WSN can be separated into flat-based and hierarchical-based routing algorithms. In flat-based routing schemes all sensor nodes have the same role. This means that every node in the network is responsible for sensing data but also for routing data packets from other sensor nodes in the network. In hierarchical-based or cluster-based routing on the other hand, the sensor nodes in the network can have different roles. For the basic setup the nodes in the network can be separated into sensor nodes and cluster heads. While the sensor nodes are sensing data, the cluster heads are responsible to forward the data that they receive from the sensor nodes to the base station. The forwarding of the data to the base station on the cluster head can be done in a single hop or via multi hop routing. The cluster head can also combine the data it received from several nodes and just transmit the combined results to the base station, trading off accuracy for reduced energy consumption. The cluster heads in a network are either normal sensor nodes just like all the other nodes in the network or special, higher energy nodes. If the cluster head is a normal node like all the other nodes in the network, a periodic change of the cluster head can be necessary to prevent depleting the power supply of the cluster head, since the workload of the cluster head is higher than the workload of the other nodes in the network.

Quality of Service

Quality of service (QoS) is a very important topic for 802.11 networks. For WSNs, other criteria often are more important than QoS. The importance of QoS for WSNs mainly depends on the application. While the occasional drop of a message or an increased latency might not be that important for a network that takes periodic temperature measurement, it can be critical for other applications. An increased level of QoS usually also leads to increased power consumption, which can be a problem for WSNs that usually have constrained power supply. Thus the tradeoff between QoS and power consumption is of importance for the design of a routing algorithm.

2.1.2 Power Aware Routing

The power supply of WSNs is usually limited and deployment in inaccessible environments combined with often large amount of sensor nodes in the network make a replacement of the energy supplies on the deployed sensor nodes unfeasible. To be able to extend the lifetime of the sensor nodes, their power consumption has to be reduced. Wireless communication is one of biggest parts of the power consumption of a typical sensor node, far bigger than the power consumption of most of the sensors that are installed on a sensor node. In [38] different approaches for power aware routing protocols for wireless ad-hoc networks are discussed that give implications for power aware routing in WSNs. This section gives an overview over different approaches for power aware routing algorithms according to [38] and presents some existing power aware routing algorithms.

Active Energy Saving Protocols

The property that is shared amongst all active energy saving protocols is their main goal. They all try to find the routing paths in the network with minimum energy consumption, thus minimizing the needed transmission power for every packet that is transmitted in the network. The algorithms that are part of this family all try to find intermediate nodes to be able to reduce the required transmission power. As already discussed in section 2.1.1 a multi-hop routing approach can reduce the total amount of energy that is needed for the packet transmission. Each node that wants to participate in the forwarding of a packet has to determine first, if it is still able to reduce the total power consumption of the transmission.

In [15] an active energy saving protocol called PARO is proposed which reduces the transmission costs for sending data from one node to another by electing intermediate nodes that participate in the data transmission. PARO is an abbreviation for Power-Aware Routing Optimization. The goal of the routing algorithm is to minimize the transmission power that is consumed in the network. To enable this, the participating senders need to be able to dynamically adjust their transmission power. The transmission power P_k to forward a packet on a given route k can be computed as:

$$P_k = \sum_{i=0}^{N_k} (T_{i,i+1}L + T_{i+1,i}l)/C \tag{2.2}$$

In this equation, N_k is the number of hops of route k and $T_{i,j}$ is the minimum transmission power at node i so that node j, which is the next hop at route k, can still receive the packet. Since the power that is consumed during a transmission is also influenced by the amount of data that is transmitted, there are three more parameters in the equation. L is the size of the transmitted frame in bits, l the size of the acknowledgment frame in bits and C is the raw speed of the wireless channel in bits per second. Thus the goal of PARO is to find the route k that minimizes the needed transmission power P_k for sending a packet from one node in a network to another node.

Location-Aided Power-Aware Routing (LAPAR), which is proposed in [70], is a distributed routing algorithm that dynamically computes location aided local routing decisions to find near-optimal power-efficient end-to-end routes for packet forwarding. To be able to make local optimal choices, each node needs to be aware of its own position and all packets have to be marked with the location of their destination node. To be able to decide if a node s should forward a packet via a relay node r, it has to compute the relay region $R_{s,r}$:

$$R_{s,r} \equiv \{i | {d_{sr}}^n + {d_{ri}}^n \leq {d_{si}}^n, i \neq r\} \tag{2.3}$$

In this equation, d_{sr} and d_{ri} are the distances between the source node s, the relay node r and a node i and n is the power exponent for the relation between transmission range and transmission power, which is a constant that is usually between 2 and 4.

If the destination node is within this relay region, sending the packet via the relay node r is more power-efficient than directly transmitting the packet from node s to the destination node. If a node has more than one possible relay node r_k for a destination node x then the relay node r_k is chosen, which minimizes the distance sum ${d_{sr_k}}^n + {d_{r_k x}}^n$ of the distance between the sender and the relay node and the relay node and the destination.

Maximizing Network Lifetime Protocols

The last section presented routing algorithms that find the optimal route in a network and then route all packets over this route. While sending all packets via the optimal route to reduce power consumption seems like a good solution, it also has a big disadvantage. All nodes that are part of the optimal routes will have much higher power consumption than the other nodes in the network, since they have to do much more work. Thus these nodes will also run out of power far earlier than the rest of the network. The goal of network lifetime maximizing protocols is to spread the workload in the network to balance the power draining in the network. Another goal is to avoid network partitioning, which is caused by draining sensor nodes that are connecting different parts of the network. The cost of the transmission route is still of importance though, the spreading of the traffic will only be able to increase the network lifetime if the new paths do not need significantly more energy than the optimal paths. Important parameters for the balancing of the network traffic include the remaining battery life, the battery drain rate and the amount of packets each node in the network has to transmit.

In [65] a routing algorithm is presented that tries to evenly distribute the power consumption in the network while minimizing the needed transmission power. The proposed routing algorithm is called Conditional Max-Min Battery Capacity Routing (CMMBCR). The basic idea behind CMMBCR is that when there are several routes between a source node and a destination node, where all nodes along the routes have a sufficient remaining battery capacity, the route with the minimum total transmission power among this pool of routes is chosen. If there are no routes though, where the remaining battery capacity is above a chosen threshold on all nodes along the route, then the routes that include nodes with the lowest battery capacity should be avoided to extend the lifetime of these nodes and thus also the total network.

Maximum Survivability Routing (MSR), which is presented in [46], chooses the used routing paths according to the remaining battery life of the nodes that are along the route towards the destination. To be able to increase the total lifetime of the network, the nodes with the least remaining battery life should be avoided. Avoiding these nodes for routing can lead to far longer routes though, which drains the total available power of the network faster due to the increased power consumption of the longer routes. Thus it can be beneficial for the total lifetime of the network if the connectivity of a small amount of nodes is sacrificed. MSR uses the following utility function and cost function to formulate this scenario:

$$u_i = u(T_i) = 1/T_i \tag{2.4}$$

$$C_R \equiv f(u_i, i \in R) = (\sum_{i \in R} u_i^{\beta})^{1/\beta} \tag{2.5}$$

The utility u_i of a node i is the inverse of the estimate of the remaining battery life T_i of the node i. The cost function C_R of a route R is the sum of the utility of all nodes along the route weighted with a parameter $\beta \geq 1$. Among all possible routes between two nodes, MSR chooses the minimum cost route since this route has the longest life expectancy and thus preserves the connectivity of the network better than all other possible routes.

In [32] the Minimum Drain Rate (MDR) routing mechanism is proposed which includes the current energy drain rate of a node in the routing metric. The cost function C_i for sending a packet via a node i is given as the ratio between the residual battery power RBP_i of node i and the drain rate DR_i at node i:

$$C_i = RBP_i/DR_i \tag{2.6}$$

The maximum lifetime of a routing path in the network can then be determined by the minimum value of the cost function C_i of all nodes that are along this route. The optimal path between two nodes in a network is then the path with the highest maximum lifetime. When a new path is routed over a node, the energy drain rate at that node is also increased which leads to a spreading of the network traffic.

Passive Energy Saving Protocols

Sensor nodes do not only consume power while transmitting or receiving packets but also while waiting in idle mode or listening on the radio module. The amount of energy that is wasted by this can be quite significant for WSNs. The basic idea of passive energy saving protocols is to reduce the energy consumption that is wasted by nodes in idle or listening mode. This is achieved by turning off as many sensor nodes of the network as possible while still maintaining the required network connectivity. For example if the sensor nodes in a network are partitioned into cells, then it is enough if there is only one sensor node awake in each cell, all the other nodes in the cell can sleep and save energy. The decision about which nodes stay awake and which can go to sleep can either be done by each node individually or it can be managed globally for the total network.

In [69] the Geographic Adaptive Fidelity (GAF) algorithm is proposed. The proposed algorithm reduces the power consumption in a network by turning off nodes that are unnecessary to maintain a certain routing fidelity in the network, data sources and data sinks are not affected by this. GAF separates the network into small grids of routes that are equivalent from a routing point of view and then turns off parts of the nodes in each grid. To balance the energy use among the nodes in each grid, all nodes are periodically turned on again to check if they should replace the current active nodes. While the GAF algorithm is independent of the underlying routing algorithm; each node needs to know its exact position to be able to find the grids of equivalent nodes in the network.

In [8] another routing algorithm is proposed that reduces the total energy consumption of the network by turning off parts of the nodes in the network, the proposed algorithm is called Span. Span does not need any localization information though, each node uses a distributed, randomized algorithm to decide whether it shall go to sleep or stay awake and act as a coordinator that is routing packets. To make a decision if a node should act as a coordinator or not, each node keeps track of all its neighbors and coordinators and the neighbors and coordinators of each of its own neighbors. Combined with the amount of energy that is available to a node, this information is used to periodically decide if a node should switch from sleep mode to coordinator mode or not, which leads to a rotation over time among the nodes of the coordinator role.

Topology Control Protocols

The topology of a WSN, which is defined as the set of possible connections between the nodes in a network, has a major influence in the possible routing paths. This also shows the downside of decreasing the transmission power of the sensor nodes too far, which can lead to a very sparse network topology and thus also very few possible ways to transmit packets. But also a too dense network has its disadvantages since it leads to far more energy loss due to packet collisions and overhearing. The goal of topology control protocols is to reduce the transmission power of the sensor nodes only so far that the desired network topology can still be achieved. The topology control is either done by each sensor node in the network for itself or is done for the total network by a global manager like the base station.

In [39] the Small Minimum-Energy Communication Network (SMECN) routing algorithm is proposed which is based on the Minimum-Energy Communication Network (MECN) algorithm that was proposed in [58]. For MECN it is assumed that two nodes in a network G' form an edge if they are able to communicate with each other with a transmission power that is below or equal to the maximum transmission power. The goal of MECN is to find a subgraph G of the graph G' that contains all the nodes of G' where all nodes that could communicate in G' can still communicate in G' but G only contains those edges of G' so that for each node the power to communicate with its neighbors is minimal. The idea of SMECN is to compute a subnetwork for a given communication network, that contains a minimum-energy path for every pair of nodes that were also connected in the original network, that allows to transmit messages between those nodes with a minimum use of energy. The subgraph produced by SMECN is smaller than the subgraph by MECN and thus also has a lower link maintenance cost which leads to savings in the energy usage. Both algorithms assume the presence of a GPS module on all nodes to be able to determine their position which is needed for the algorithms to work.

Energy-Efficient Multicasting and Broadcasting Protocols

Multicasting as well as broadcasting are popular for wireless networks since they allow a fast spreading of information to multiple network participants due to the broadcasting nature of the wireless medium. The general idea behind energy-efficient multicasting or broadcasting protocols is to build a transmission graph that is able to minimize the energy required for transmissions while still reaching all desired receivers. The use of such algorithms is only viable though if the energy savings in comparison to standard flooding algorithms are higher than the energy that is required to create and maintain the minimum energy transmission graph. This also means that energy-efficient multicasting and broadcasting algorithms are only useful in WSNs with a high amount of data that is transmitted via broadcasting or multicasting. In networks with high mobility it can also be unfeasible to use those algorithms since the required frequent updates will counteract the energy savings of the routing algorithm.

An energy-efficient broadcasting algorithm that is an approximation to the NP-complete problem of the minimum-energy broadcast tree is proposed in [41]. The problem of finding a minimum-energy broadcast tree is reduced to the problem of finding a minimum-energy broadcast tree of an auxiliary graph. The approximate solution of the optimization problem for the auxiliary graph is then used to find an approximate solution for the original problem. The proposed algorithm is able to find an approximate solution within a bounded performance. In [41] also an energy-efficient multicasting algorithm is proposed that is able to find an approximate solution to the NP-complete problem of finding a minimum-cost multihop tree, the approach is similar to the approach for the minimum-energy broadcast tree.

A different approach for energy-efficient broadcasting is proposed in [67]. The proposed algorithm is called Broadcast Incremental Power (BIP) which builds multi-hop broadcasting trees with the goal to minimize the needed transmission power. The algorithm builds the broadcasting tree incrementally, after each node that is added to the broadcasting tree it is evaluated if it is cheaper to increase the transmission power on a node or to start broadcasting on another node to reach the next node in the network. BIP does not guarantee that the resulting broadcasting tree is optimal, but this tradeoff has to be made to keep the algorithm scalable. Based on BIP a multicasting algorithm is also proposed in [67] which is called Multicast Incremental Power (MIP). MIP starts its operation from the broadcasting tree created by BIP and prunes it down since only the intended receivers of the multicast packet need to be reached by the multicast tree.

2.1.3 Energy Harvesting Aware Routing

The last part presented approaches and algorithms for power aware routing. But when the WSN is powered with an energy harvesting device these algorithms are no longer optimal since the requirements on the routing algorithms change. In contrast to power aware routing the goal of EHAR is not to keep the power consumption that is used for communication as low as possible.

Even when just using power aware routing algorithms, the usage of EHDs in addition to the normal power supply will increase the total lifetime of the network. But energy harvesting can not only be used to extend the lifetime provided by the battery. When the energy consumption stays below the harvested energy, it is theoretically possible to infinitely increase the lifetime of the network. Thus the goal for EHAR is to provide the best possible routing without exceeding the amount of energy provided by the EHD over a given period of time.

For example when a sensor node is equipped with a solar panel as energy harvesting device, the solar panel will have a typical power curve over the period of a day. The energy that is provided by the EHD will not be the same for every day though. The difference between a sunny and a cloudy day for example will be quite big and results in a completely different power curve. While a normal battery power supply can compensate the fluctuations in the power provided by the EHD, this is not an optimal solution since the power supply will be drained sooner or later. Rather, the algorithm should be able to adapt to the changing amounts of energy provided by the EHD.

Energy Harvesting Aware Routing by Kansal et al.

In [61] an energy aware routing algorithm was proposed. The algorithm uses probabilistic forwarding to send traffic on different routes. The cost metric that is used for determining the probability of the different routes has already been proposed in [7]. The cost metric consists of the energy that is used to transmit and receive on the link and the residual energy of the receiver node. While the proposed algorithm does take the residual energy into account when determining the route in the network, the algorithm is not energy harvesting aware. In [28] and [29] an energy harvesting aware routing algorithm is proposed that is based on the algorithm presented in [61], but with a cost metric that takes the harvested energy of the sensor nodes into account.

Contrary to other routing algorithms, the proposed algorithm does not only store routing information of the node that is best suited according to a given metric for routing the packet towards its intended receiver. For the proposed algorithm each node stores the routing information for all nodes that are as close as or closer to the base station then the node itself. Each time a packet is transmitted, the node then chooses one of the routing table entries as the next hop, the probability for each entry to be chosen depends on the chosen cost metric. With the right cost metric the algorithm is able to determine the next hop according to the energy harvesting information but the algorithm also guarantees that not all messages are always routed over the node that has the best value for the chosen cost metric.

A cost metric that is taking the energy harvested by an EHD into account has been proposed in [28]. The proposed formula to determine the energy potential of a sensor node is:

$$E_i = \omega * \rho_i + (1 - \omega) * B_i \tag{2.7}$$

The energy potential E_i is determined as the weighted sum of two parameters. The parameter ρ_i is value for the expected rate of energy harvesting on the sensor node. The second parameter B_i represents the residual battery level. The weight parameter ω is used to set the ratio of the factor of influence of the energy harvesting rate and the residual battery level. The value range of ω is $0 \leq \omega \leq 1$, typically the value of ω is set close to 1. This means that the focus is set on the expected energy harvesting rate and the battery should only be used to compensate for fluctuations of the harvesting rate. A low value of ω might be useful in cases, where the EHD is not able to sustain the workload of the sensor node and is just used to increase the lifetime of the sensor node. A value of ω of 0 simulates a case without an EHD, then the algorithm is a normal power aware routing algorithm. When ω is set to 1, the algorithm simulates the case when no battery is present and the only available energy is the energy gathered by the EHD. This also means that there is no power available from the battery to compensate for fluctuations of the energy gathered by the EHD.

To determine the probabilities of the available routes on a sensor node, the costs for all the outgoing links of the sensor node have to be calculated. The cost is calculated as the inverse of the energy potential of the node on the receiving side of the transmission. The cost is the same for all links that are entering the same node since the cost only depends on the energy level of this node. This means that in a directed graph where every node is represented as a vertex ν_i and each possible hop between any pair of nodes i and j is represented as an edge e_{ij}, the cost for each edge c_{ki} that is entering node i can be written as [28]:

$$c_{ki}(e_{ki}) = 1/E_i \ \forall k \in \{k | e_{ki} \in E_{comm}\} \tag{2.8}$$

In this equation E_{comm} represents the total amount of feasible wireless connections. The amount of feasible connections depends on the topology and the density of the network and the characteristics of the radio module of the sensor nodes.

The formula to determine the probability of an outgoing connection was already proposed in [61]. The formula for the probability is:

$$P_{ij} = \frac{1/c_{ij}}{\sum\limits_k 1/c_{ik}} \tag{2.9}$$

The probability P_{ij} is determined by the inverse of the edge cost c_{ij} of the connection between the nodes i and j divided by the sum of the inverse of the edge costs of all outgoing connections. Since the probability of a node to be chosen depends directly on the energy level of the node, the algorithm is able to spread the traffic across the network according to the energy that is available to the sensor nodes. The algorithm takes only the energy level of the next hop into account though. Consider two outgoing multi-hop connections from a node to the base station. The first connection has a node with a high energy level at the start and only nodes with a low energy level afterwards; the second connection consists only of nodes with a medium energy level. In the proposed algorithm, the second connection will have a lower probability even though it is the favorable connection.

Other Work on Energy Harvesting Aware Routing

In [35] Lattanzi et al. present a methodology for the evaluation of the applicability of routing algorithms for environmentally powered WSNs and use this methodology to compare several routing algorithms. For the comparison of different routing algorithms the ratio between the maximum energetically sustainable workload of a routing algorithm and the optimum maximum energetically sustainable workload has to be determined. The maximum energetically sustainable workload depends not only on the routing algorithm but also on the network setup and the environmental power constraints.

The maximum energetically sustainable workload is defined as the workload that can be energetically sustained by every node in the network that is involved in the packet processing and routing and that cannot be increased without violating the energy sustainability on one or more nodes. The theoretical optimum maximum energetically sustainable workload is given by the best routing algorithm applicable for the current setup and can be computed with an extended version of the Ford Fulkerson maxflow algorithm presented in [5].

The performance of several routing algorithms is compared for several network setups with different power maps. The by far best performing algorithm in all setups is the randomized maxflow algorithm that uses offline calculated routing tables to achieve the same flow that is used to determine the optimum maximum energetically sustainable workload. The algorithm with the best results without predetermined routing tables is randomized minimum path recovery time. Here the path selection on a node is done randomly; the probability of a path depends on the cumulative recovery time of the path. The cumulative recovery time is the sum of the recovery times of each edge of the path. The workload that is achieved by this algorithm is only between 10% and 40% of the optimum workload though. An interesting discovery is that the randomized maxflow algorithm with routing tables that are calculated for the inverse power map still outperforms a minimum path routing algorithm that always chooses the route with the least hops.

In [33] a routing algorithm is presented that implements a maxflow algorithm similar to the one used to determine the optimum maximum energetically sustainable workflow in [35]. Since the network should be able to adapt to changing power conditions the network has to be able to calculate the maxflow. To calculate the maxflow the Push-Relabel algorithm [9] is used in a distributed implementation. One assumption that is made to simplify the calculations is that the transmission power is not dynamically adapted to the actual distance of the receiver. This assumption, that holds true for many real world implementations of WSNs, means that all outgoing connections from a node will have the same cost. The biggest downside of the presented algorithm is that the operations and data transmissions for the periodic recomputation of the maximum energetically sustainable workload cost energy. This means that there is always a tradeoff between the energy needed for the recomputations and the optimality of the routing.

In [60] a different implementation of an algorithm that is based on finding the optimum maximum energetically sustainable workload is presented. They propose an optimum maxflow routing (OMFR) algorithm that is optimally using the available environmental power and is able to adapt to time-varying environmental conditions. For the ideal OMFR algorithm some simplifications have to be made. The ideal algorithm assumes that all packets have the same size of one bandwidth unit, that each packet is handled at a time and that each router always has updated information of the best path according to the residual capacity of the nodes. In [60] it is shown though that these assumptions are not needed for the algorithm to work. The violation of the first two assumptions does not lead to a really different behavior. When the third assumption is no longer true and the routing information is only updated periodically and not after every packet the nodes can no longer choose the optimal path for every packet. Rather each node will use the same path for each packet during one update period. While this will lead to imbalanced energy consumption during one period, it will even out over time as long as the energy supply of the nodes can power the transmissions occurring during one period. Similar to the routing maxflow routing algorithm presented by [33] the OMFR algorithm also offers a tradeoff between optimality of the routing algorithm and frequency of the updates of the routing information.

Since just finding the most energy efficient route in the network is not sufficient for energy harvesting WSNs in [25] a distributed energy harvesting aware routing (DEHAR) algorithm is proposed that is able to adapt to changing energy distributions in the network. The algorithm that is designed for multi-hop networks with a single data sink is based on a shortest path routing algorithm. But instead of just using the path length as routing metric a distance penalty is added for each hop on the path. The distance penalty of a hop depends on the available energy at the hop. Thus the new routing metric is defined as the energy distance. With the use of energy distances as routing metric the routing algorithm is able to find high energy paths even if they have a far higher path length then the shortest path to the data sink. The increased cost of transmitting packets to the data source is traded in here for the higher energy level of the path. To be able to keep information on the energy distances of all possible paths up to date on all nodes in the network, every node has to inform its neighbors if its distance penalty grows or shrinks significantly. When a node finds the current best route towards the data sink it will transmit all its packets over that route until it is no longer the optimal route due to changes in the distance penalties. While the algorithm is able to find good routes even between several low energy areas in a network, the algorithm relies on frequent updates of the distance penalties to effectively spread the occurring traffic over more than the nodes on a single path towards the data sink.

In [34] an EHAR scheme that aims at minimizing the latency of the transmitted packets is proposed. The algorithm assumes that each node in the network uses duty cycling (see section 2.3.1) where each node chooses its duty cycle according to the harvested energy to guarantee a perpetual operation of the node. The network is organized as a random graph, this means that each node can route data to the sink node over different routes if there is more than one available. The algorithm chooses the next hop not according to any energy based criteria but chooses the node for the next hop that offers the minimum latency. While the node that wakes up first would minimize the latency for a single hop, the situation is not that easy for multiple hops. A metric is proposed that chooses the next hop for a packet according to the latency of the first hop and an estimated latency of the remaining hops to the data sink. Nodes with a shorter duty cycle period will most likely have more data to route, but a shorter duty cycle period also means that the node has more energy available than others. Thus this algorithm can also lead to a spread of the network traffic according to the harvested energy.

The routing algorithm proposed in [27] is specifically designed for WSNs that are equipped with a hybrid energy storage system (HESS) that consists of a supercapacitor and a rechargeable battery. The battery can only be recharged several hundred times but it can be used for long time energy storage since the power leakage is low. The supercapacitor can be recharged a million times but has a high power leakage; its stored energy should thus be used first. The proposed algorithm is called Communication Using Hybrid Energy Storage System (CHESS). The selection of a route with CHESS is made according to the sum of CHESS metrics of all nodes of the route, the route with the lowest sum value being the best. The CHESS metric depends on the energy level of the supercapacitor and the rechargeable battery. If the supercapacitor has enough energy to route a packet then the value of the metric will be zero. If this is not the case then the metric will be set to a value greater zero according to the energy level of the rechargeable battery.

Most proposed routing algorithms are always evaluated under perfect conditions in a network without noise and packet collisions. Furthermore, the cost of keeping the routing metric updated on all nodes of the network at all times is usually ignored. In [20] three EHAR algorithms are compared under real world conditions to evaluate the impact on the routing algorithms. The first used routing algorithm is the Energy-opportunistic Weighted Minimum Energy (E-WME) algorithm that is proposed in [42]. The other two algorithms that are used are randomized maxflow and randomized minimum path recovery time. They are both proposed in [35] and were already presented at the beginning of this section. The real world conditions that are evaluated are packet collisions, low power MAC protocols, a realistic wireless channel, the protocol overhead and different energy harvesting scenarios. However, the low power MAC protocol that is used for the evaluation does not provide any mechanism to avoid packet collisions, which leads to a higher packet collision rate than necessary. The evaluation shows that a modified version of the randomized minimum path recovery time deals best with the real world conditions while especially the randomized maxflow algorithm suffers from very high packet loss rates under real world conditions. In general, the paper shows that good performance of a routing algorithm under perfect conditions does not guarantee a good performance of the same routing algorithm under realistic network conditions.

2.2 Network Coding

NC is a hot topic for wired and wireless networks at the moment. In the first part this section presents the concept of NC. The second part then looks at problems and already existing implementations of NC for WSNs. The third part presents the idea of ONC.

2.2.1 Basics

NC is a rather new research topic, it was first mentioned in [2] in the year 2000. The first proposal intended NC to increase multicast throughput in networks and evolved from the topics of diversity coding and source coding. The idea was to increase the possible throughput of the network, by combining packets into a single packet with the same length and reconstructing the original packets again at the receiver side. It is important to note that when the reconstruction is successful, the data is exactly the same as at the start, there is no sort data aggregation done.

With the use of NC a step away is made from the way the information flow in networks was handled until then. All routing schemes so far always left the data part of the packets untouched. Only on higher network layers methods such as data aggregation tried to combine the data of packets, but were always accompanied by a loss of information. By using NC different information flows no longer only share the same network resources, it allows to move from separate information flows to combined flows that permit better utilization of the network resources.

Combining and restoring of messages adds some additional computations to the routing. The use of linear NC allows reducing the computational overhead for using NC. Here the combining of data, represented as numbers in a finite field, is done by using simple linear combinations only. [40] shows that linear NC is enough to achieve the max flow that is possible, there is no need for higher order arithmetic during the combining and the restoring of the packets. The linear combination of the packets also allows that the packet length stays the same. If the packets to combine are not all of the same length, the combined packet will be as long as the longest packet, all shorter packets will be filled up with zeros before the packets are combined.

In [11] the formulas that are needed for the implementation of NC are explained. The operations for NC are all done in a finite field. For linear NC encoding and decoding is done in the finite field $\mathbb{F}_{2^s}$. The s in the definition of the field represents the used symbol size in bits. The symbol size is the amount of bits that are encoded in one operation. The amount of different symbols that exist in a finite field is called the order. The order of the given finite field is 2^s. For a symbol size of only one bit, the finite field consists only of the two elements 0 and 1.

To encode a number of packets $M^1, ..., M^i, ..., M^n$ each of the packets is combined with a sequence of coefficients $g_1, ..., g_i, ..., g_n$ in $\mathbb{F}_{2^s}$. The encoding of the messages is done on a per symbol basis, the formula for combining one symbol of the combined message is:

$$X_k = \sum_{i=1}^{n} g_i M_k^i \tag{2.10}$$

In this formula, X_k and M_k^i represent the k-th symbol of the corresponding messages X and M^i. To be able to decode the message again, both the encoding vector that contains the coefficients $g_1, ..., g_i, ..., g_n$ and the information vector X are needed. Thus either the encoding vector has to be transmitted additionally to the information vector or each sensor node has to use an encoding vector that is fixed. It is not only possible to encode normal messages, but also encoded packets can be encoded again. While the encoding of the message is just the same, the difference when encoding an already encoded packet again lies in the encoding vector. The new encoding vector is generated by encoding the original encoding vector with the encoding vector that was used for the encoding of the already encoded packet.

To be able to read the data of a message, the receiver has to restore the original message again. Let's assume that the node that wants to restore a message has received the set of encoded packets $(g^1, X^1), ..., (g^i, X^i), ..., (g^m, X^m)$. To be able to recover the original packets, the following set of equations has to be solved:

$$X^j = \sum_{j=1}^{n} g_i^j M^i \tag{2.11}$$

The unknown variables in the equation that shall be discovered are the M^i. The amount of equations in the linear system is m while the amount of unknown variables is n. To have a chance of solving the system of equations the inequality $m \geq n$ has to be complied. This means that the amount of messages that a node needs to receive to restore the original messages is at least as high as the number of original messages. This requirement is only sufficient though if all of the equations are not linearly dependent. This can be easily fulfilled though by choosing the encoding vectors accordingly.

There exist several possible fields of problems that can benefit from the use of NC [11]. The most prominent application for NC is the throughput increase in both wired and wireless networks. Figure 2.1 shows a simple butterfly network configuration where the two sensor nodes A and B both want to transmit data to the sensor nodes E and F. Without NC the sensor nodes C and D are a bottleneck in the network since they have to transmit twice as many packets as the other nodes in the network. With the use of NC, the node C can combine the two packets and just send out one combined

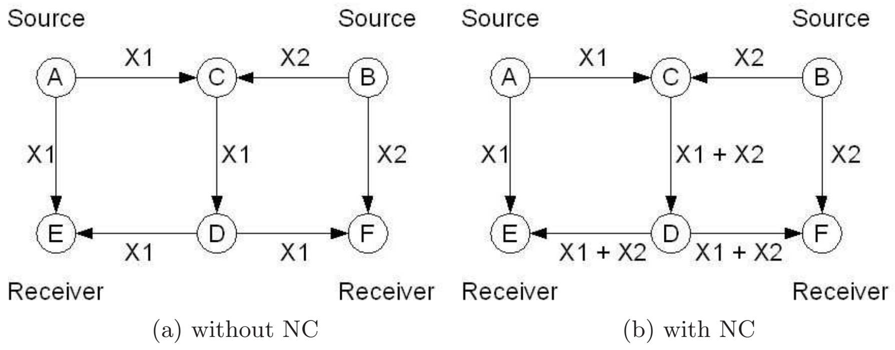


Figure 2.1: Simple network setup without and with NC

packet, the original messages can then be restored again on the nodes E and F. Thus, it is possible to theoretically double the throughput of the presented network. Another possible application for NC is data distribution. With the use of NC it is possible to decrease the amount of packets that are needed to transmit large amounts of data to many nodes at the same time, as is the case in a peer-to-peer network for example. This is achieved by sending out random linear combinations of the data instead of each part of the data alone. Further applications for NC include reliable data transmission, network monitoring, securing of data transmissions against eavesdropping and protection against modified packets.

2.2.2 Network Coding in WSNs

Most of the research on NC is done for wired networks and wireless networks of the 802.11 family but NC also has some interesting applications for WSNs. This section presents an overview on already existing approaches for the use of NC in WSNs.

One of the main uses of NC in WSNs is multipath routing which can be used in WSNs to either increase the resilience or the bandwidth of the connection between a source and a destination node [51]. In [66] NC is used to find a low cost solution to the construction of disjoint multipath routes. The proposed efficient multipath routing approach combines directed diffusion [24] and randomized NC [22] to achieve a low cost solution for finding disjoint multipath routes in a network. [62] proposes an energy efficient multipath routing algorithm that uses NC to reduce the necessary amount of paths. This allows decreasing the energy consumption of multipath routing while still achieving the same reliability. But to achieve this the transmitted data needs to be extended with some bytes of metadata and also a small computational overhead is caused by the algorithm. Also in [16] NC is used reducing the amount of necessary packet transmissions in a multipath sensor network. The network that the algorithm is designed for is not a normal WSN though, the algorithm is designed to counter the high error rates that are present in underwater sensor networks. When using WSNs for medical applications, reliability of the data communication is of great importance. But since the WSNs for medical applications are usually carried on the body, the size and thus also the available power is usually quite limited. [47] propose the use of NC to be able to cost efficiently increase the reliability of the network by adding redundancy.

In [68] the use of NC is proposed for intra-cluster information exchange. They propose the use of NC for scenarios, where each of the nodes in a cluster is a data source that wants to transmit its data to all other nodes in the network like it can be the case during route discovery or updating or when congestion control information is distributed inside the cluster. The use of NC allows to reduce the amount of necessary transmissions for this kind of all-to-all broadcasting scenarios. [53] also proposes the use of NC to reduce the amount of packets that are needed for all-to-all broadcasts in a network. All-to-all broadcasts are usually implemented as store and forward where each node first stores the packet that it received and then broadcasts it again over the radio interface. With the use of NC it is possible to combine the received packet with packets that are overheard from neighboring nodes to reduce the total amount of broadcasts that are needed.

A completely different use for NC in WSNs is shown in[4]. They propose the Location-aware Network Coding Security (LNCS) protocol that provides security services like data confidentiality, authenticity and availability with the use of NC. LNCS divides the network into cells, where nodes of a cell are close to each other. If an event occurs in a cell aggregated information from the nodes in the cell will be partitioned using a secret sharing algorithm and forwarded to the receiver. Random NC is used to generate redundant information which makes it possible to recover from packets that are lost by the radio or dropped by malicious nodes.

2.2.3 Opportunistic Network Coding

So far the research on NC was always focused on scenarios where it was always clear where and which packets would be combined. ONC tries to move away from this scheme and introduce NC to normal network communication where the decision if NC can be used or not depends on the actual traffic. The idea of ONC is to exploit the broadcasting nature of the wireless medium to detect coding opportunities and use them to forward multiple independent packets with a single transmission [31].

One of the first algorithms that proposed the idea of ONC was COPE [31]. COPE was developed for the use in 802.11 wireless networks and is operating between the MAC layer and the IP layer. The mechanics of COPE can generally be separated into three main techniques, opportunistic listening, opportunistic coding and the learning of the state of the neighboring nodes.

For the first task which is opportunistic listening, all nodes have to be in promiscuous mode. The COPE algorithm then tries to receive all possible packets that it can hear on the radio and stores the overheard packets for a limited period of time. To inform the other nodes in the network on the packets that a node was able to overhear, the nodes broadcast reception reports. Those reports are either attached to normal data packets or periodically transmitted on their own if no other data packets are sent.

The second task is opportunistic coding which focuses on deciding about which packets are encoded. The goal is to maximize the number of packets that are transmitted with a single transmission while ensuring that all intended next hops have enough information that they can recover the packet that was intended for them. When a node wants to combine three packets that are intended for different receivers this means that each of the receivers needs to have the other two packets to be able to recover the packet that was intended for this receiver. Or in general, if there are n packets combined then each of the n receivers needs to have the other $n - 1$ packets to be able to decode his native packet. Thus the goal of ONC is to find the maximal possible n that still allows all receivers to decode their packets.

The third task is the learning of the neighbor states. This is very important since without knowing which packets are available to the other nodes in range, the node cannot decide which packets it is allowed to combine. As already mentioned before, reception reports tell the other nodes which packets a node received. A node can not only rely on reception reports though, since they can get lost due to congestion when there is high traffic or they might arrive too late when the traffic is low. When a node does not know for sure whether a node was able to receive a certain packet or not, it has to guess. To enable an intelligent guessing, each node computes and broadcasts the delivery probability that is computed by wireless routing protocols, for each of its links. COPE uses these probabilities to estimate whether a node was able to receive a packet or not. In the occasional case that a wrong guess was made and one of the next hops is unable to decode the packet only the native packet for that node has to be retransmitted.

One big downside of the COPE algorithm is that the coding is limited to only one hop. Each encoded packet is decoded again at the next hop and the node retrieves its native packet which might get encoded again with different packets for the next hop. There is no transmission of encoded packets over more than one hop. This means that a lot of computational overhead is needed on every hop for encoding and decoding. Another downside of COPE is its dependency on the data routes that are chosen outside the domain of COPE. To find ways around these limitations, [36] proposes DCAR (Distributed Coding-Aware Routing). DCAR offers mechanisms for the discovery of paths between source and destination nodes and also for the detection of opportunities for NC over much wider parts of the network. This allows DCAR to find specific routes in the network that offer a high throughput with the use of NC that would not have been found by conventional wireless routing protocols that do not consider the use of NC.

2.3 Node Power Saving Strategies

The previous two sections presented routing algorithms with the goal of increasing the lifetime of the total network. The approaches presented in this section try to reduce the energy consumption of the single nodes and thus increasing the network lifetime. This section presents the concept of duty cycling in the first part. The second part then presents the concept of low power listening, which is a special case of duty cycling.

2.3.1 Duty Cycling

One of the biggest problems of WSNs is their limited power supply. While the previous chapters proposed networking strategies to reduce the power consumption in the network, there are also possibilities to reduce the power consumption on the nodes. Even if a node currently has nothing to do and is just operating in idle mode, some modules of the node will still consume a considerable amount of power. The basic idea behind duty cycling is to periodically turn modules on and off to reduce the power that is wasted while there is nothing to do and thus also reduce the average power consumption.

The possible energy savings with duty cycling depend on the relation between the on time and the off time. In [10] an overview is given on the formulas that are relevant for duty cycling. The period of a duty cycle T_{DC} is given by the sum of the on time T_{on} and the off time T_{off}.

$$T_{DC} = T_{on} + T_{off} \tag{2.12}$$

The duty cycle DC is defined by the ratio of the on time T_{on} and the period of the duty cycle T_{DC}.

$$DC = \frac{T_{on}}{T_{on} + T_{off}} \tag{2.13}$$

The values for T_{on} and for T_{off} cannot be chosen completely free, there are some constraints. For the minimum value of the on time T_{on} the startup latency $T_{startup}$ of the module has to be considered. Additionally the module should be turned on at least for the minimum task time T_{task} that the module needs to perform its task, for example the time needed to read the temperature when duty cycling the temperature sensor. The on time T_{on} needs to be at least as long as the sum of those two.

$$T_{on} \geq T_{startup} + T_{task} \tag{2.14}$$

The off time T_{off} is determined by the on time T_{on} and the chosen duty cycle. The computation of the reduced averaged power P_{avg} of the module can be calculated by averaging the needed power over a duty cycle period T_{DC}.

$$Power_{avg} = \frac{1}{T_{DC}} \int_{T_{DC}} Power(t)dt \tag{2.15}$$

Another big influence on the energy savings that are made possible by duty cycling is the ratio between the power that is consumed by a turned on module and the power that is still consumed by the module when it is turned off.

For battery powered WSNs the selection of the desired duty cycle depends mainly on the minimum duty cycle value that still allows the application that is running on the sensor nodes to fulfill its job. The chosen duty cycle might be a bit above the minimum value to be able to handle some unexpected events but other than that any increase of the duty cycle leads to a reduced lifetime of the sensor nodes. For energy harvesting powered WSNs the situation is a bit different though. The optimal duty cycle here is one that allows the node to consume exactly the amount of energy provided by the energy harvesting device thus achieving energy neutral operation.

Determining the Maximum Sustainable Duty Cycle

To be able to determine the duty cycle for a sensor node that is sustainable with the power that is provided by the EHD, the power output of the EHD and the power consumption of the sensor node have to be characterized first, a model for this characterization is presented in [30]. The power output of the EHD of a sensor node at time t is given as $P_s(t)$ and the power that is consumed of the sensor node at time t is given as $P_c(t)$. To allow energy neutral operation, the consumed power has to be lower than or equal to the harvested power. Thus the condition for energy neutral operation without any energy storage buffer is given by:

$$P_c(t) \leq P_s(t) \tag{2.16}$$

An energy buffer is useful for energy harvesting though since else all harvested energy that is not needed to fulfill equation 2.16 is wasted. When adding an energy storage buffer to the condition of energy neutral operation, the characteristics of the energy buffer need to be considered too since an energy buffer is never optimal. Just like a battery an energy storage buffer can be characterized by its charging efficiency η, its leakage power p_{leak} and its capacity limit B. With these parameters of the used energy storage buffer the condition for energy neutral operation is given by:

$$B_0 + \eta \int_0^T [P_s(t) - P_c(t)]^+ dt - \int_0^T [P_c(t) - P_s(t)]^+ dt - \int_0^T P_{leak}(t) dt \geq 0 \ \forall \ T \in [0, \infty) \tag{2.17}$$

In this equation B_0 denotes the initial energy level of the energy storage buffer at time $t \leq 0$ and $[x]^+$ is a rectifier function with $[x]^+ = x$ for $x > 0$ and 0 for $x \leq 0$. Equation 2.17 does not consider the capacity limit of the used energy buffer though. If the value on the left side of the equation exceeds the capacity limit B of the energy storage buffer, the excess energy will be dissipated as heat. This excess energy needs to be added to the right hand side of equation 2.17 to correct the criteria for energy neutral operation.

The power consumption of an active sensor node will in almost all cases be above the maximum possible value for $P_c(t)$. The goal now is to find a duty cycle that allows a sensor node to stay in energy neutral operation. The chosen duty cycle is also relevant for the utility that an application on a sensor node can offer. If the duty cycle is too low, the application will not be able to do its job and will produce no output and thus be useless. There is also an upper bound for the duty cycle where a longer active period will no longer increase the utility of an application since all work can get fully completed during each period. In [30] [23] the utility of an application $U(D)$ that is depending on the duty cycle D is defined by:

$$U(D) = 0 \quad \text{if} \quad D < D_{min} \tag{2.18}$$

$$U(D) = k_1 + k_2 D \quad \text{if} \quad D_{min} \leq D \leq D_{max} \tag{2.19}$$

$$U(D) = k_3 \quad \text{if} \quad D > D_{max} \tag{2.20}$$

When the duty cycle is below the lower bound for the duty cycle D_{min} the utility of the application is 0. If the duty cycle is higher than the upper bound for the duty cycle D_{max} the utility of the application is at the maximum value k_3 and a higher value for the duty cycle will no longer increase the utility. If the duty cycle is between the two bounds then the utility of the application is given by the two constants k_1 and k_2 that define the initial utility at D_{min} and the rise of the utility value as a function of the duty cycle.

Since the energy that is provided by the EHD is continuously changing, an optimal duty cycling algorithm must be able to adapt the duty cycle according to the provided power. The goal is to choose the duty cycle $D(i)$ for each time slot $i \in \{1, ..., N_w\}$ of a periodic time window N_w to maximize the total utility $U(D)$ over the total period of time. The harvested power in a time slot is given by $P_s(i)$ while the power consumption of the active node is given by P_c, the power consumption in sleep mode is close to 0 and is ignored. The residual energy of the energy storage at the start of a time slot is given by $B(i)$, the residual energy at the end of the time slot is thus $B(i+1)$. The battery used in any time slot i can then by calculated as:

$$B(i) - B(i-1) = \Delta T D(i)[P_c - P_s(i)]^+ - \eta \Delta T P_s(i)\{1 - D(i)\} - \eta \Delta T D(i)[P_s(i) - P_c)]^+ \tag{2.21}$$

For an energy neutral operation the battery level $B(N_w)$ at the end of the time window N_w needs to be at least as high as the battery level $B(1)$ at the start of the time window. With the knowledge of the complete energy availability profile of a sensor node including future energy values it is possible to calculate the optimal duty cycle. Finding the optimal duty cycles $D(i)$ for each time slot is done by solving the following optimization problem:

$$max \sum_{i=1}^{N_w} D(i) \tag{2.22}$$

$$\begin{gathered} B(1) = B_0 \\ B(N_w + 1) \geq B_0 \\ D_{min} \leq D(i) \leq D_{max} \ \forall i \in \{1, ..., N_w\} \end{gathered} \tag{2.23}$$

The initial energy level of the battery is given by B_0, all further battery levels are calculated according to equation 2.21. Since the energy of future time slots is not available in real implementations, a duty cycling algorithm has to be able to predict the future harvested energy and adapt to unexpected changes.

The harvesting-aware power management that is proposed in [30] [23] consists of three parts, the energy generation model, the calculation of the optimal duty cycle based on the predicted energy and the real time adaption of the duty cycle to observed changes in the energy generation profile.

The prediction of future harvested energy by the energy generation model is based on Exponentially Weighted Moving-Average (EWMA) that is able to adapt to changes in the profile of the harvested energy. The number of historic values that are needed to calculate the average is given by the number of slots w that the time window N_w is split into. The time window for a solar panel for example would be 24 hours; the amount of time slots depends on the desired duration of the time slots. The saved historic values are an average of all historic observations in that time slot and are calculated according to:

$$\overline{x}(i) = \alpha \overline{x}(i-1) + (1-\alpha)x(i) \tag{2.24}$$

The historical average for slot i is given by $\overline{x}(i)$, the generated energy for slot i is given by $x(i)$ and α is a weighting factor that determines the influence of current generated energy on the historical average.

The calculation of the optimal duty cycle for the next time window is based on the predicted energy values for the slots of the time window N_w that were calculated by the energy generation model. These values are used to solve the optimization problem that was defined in equation 2.22 and 2.23. The results from the optimization problem are used as duty cycle values for the next time window. While the calculated duty cycle values are optimal for the predicted energy values, the real energy values that are observed during a time slot can differ greatly from the predicted value.

The deviation between the predicted harvesting power level $P_s(i)$ and the observed power level $P'_s(i)$ necessitate a real time adaption of the duty cycle. The real time adaption is needed both to prevent the sensor nodes from running out of energy if the real values are lower than the predicted values and to make use of any additionally available energy if the real values are higher than the predicted ones. The actual difference in available energy also depends on whether the consumed power is higher than the harvested power or not. This excess energy X for a time slot i can be calculated by:

$$X = \begin{cases} P_s(i) - P'_s(i) & \text{if } P'_s(i) > P_c \\ P_s(i) - P'_s(i) - D(i)[P_s(i) - P'_s(i)](1 - \frac{1}{\eta}) & \text{if } P'_s(i) \leq P_c \end{cases} \tag{2.25}$$

If the excess energy X is below 0 in a time slot i then the energy that is available at the end of the time slot is lower than the predicted value. To remain energy neutral, the duty cycles of future time slots need to be lowered. To reduce the losses by the non-ideal energy storage, the duty cycles of the time slots with the lowest predicted power levels $P_s(i)$ should be lowered first. If the excess energy X is greater than 0 this means that the duty cycles of future time slots can be increased. Here the duty cycles of the time slots with the lowest consumed power P_c should be increased first to maximize the total throughput.

In [50] an energy harvesting aware power management algorithm is proposed that adapts the energy consumption of a sensor node according to the expected harvested energy. The proposed algorithm consists of two software tasks, an estimator to predict the future harvested energy and a controller to adapt application parameters based on the results of the estimator. One difference of the proposed algorithm to other work on the topic is that all harvested energy is always stored first; a direct consumption of the harvested energy bypassing the energy storage would require a change in the system concept.

To be able to estimate future energy values the used power model has to be defined first. For modeling the time is separated into time slots t with duration T. The energy from the EHD is not modeled as a continuous function, since sensing is only done at the start of each time slot $t \in \mathbb{Z}_{\geq 0}$, the harvested energy in a time interval $[t, t+1)$ is given by $E_s(t)$. The estimator uses the available data of the harvested energy from all already passed time slots. With these values the estimator predicts the amount of energy that will be gathered in the future N estimation time slots L. The actually used estimation algorithm is not defined in [50]

The energy consumption of a sensor node is separated into the energy consumed by each task that is running on the sensor node. A single instance of a task τ_i needs the energy e_i from the energy storage during its execution. The time-variant rate $s_i(t)$ determines how often a task is executed during each time period T. The energy $E_i(t)$ that a task needs during an interval $[t_1, t_2)$ with $t_1, t_2 \in \mathbb{Z}_{\geq 0}$ is given by:

$$E_i(t_1, t_2) = \sum_{t_1 \leq u < t_2} e_i \cdot s_i(u) \tag{2.26}$$

Each application that is running on a sensor node consists of several tasks, the dependencies of the tasks and their activation intervals are modeled in a rate graph. The activation rates of the tasks on an application are not fixed though. The controller is responsible for adapting the activation intervals of the different tasks to optimally use the predicted available energy.

2.3.2 Low Power Listening

The radio module is usually the module of a sensor node with the highest energy consumption. There are two main reasons for that. The first is that both sending and receiving have a high energy cost. The second reason is that even when transmissions happen rarely, the radio module always has to be in listening mode to be able to receive a packet in case a transmission is happening. In 802.11 networks the cost of idle listening can be as high as 50% to 100% of the energy required for receiving according to measurements [71].

The idea of low power listening is to apply duty cycling to the radio module [55]. While the duty cycling of modules like a sensor does not necessitate any additional logic, the case is not that simple for the radio module. Since the sensor nodes still need to be able to communicate with each other, the low power listening algorithm has to ensure that both receiver and sender are awake at the same time to enable communication. To enable this, low power listening is implemented on the MAC layer. The algorithms can be put in two categories, synchronous and asynchronous approaches. The idea of the synchronous approach is to synchronize the sleep cycles of nodes that want to send data to each other to guarantee that both nodes are awake when a transmission is happening. Asynchronous algorithms on the other hand send a preamble before the actual packet to ensure that the receiver is awake when the actual packet is transmitted.

S-MAC

One of the first low power listening MAC protocols that have been specifically designed for WSNs is Sensor-MAC or short S-MAC [71]. To reduce the energy usage during the idle listening times, the scheme proposes a periodic listen and sleep cycle. To enable the nodes to transmit data to each other, S-MAC uses a synchronous approach.

Theoretically each node is able to choose its own listen and sleep schedule. But to reduce the overhead required for synchronization the nodes try to keep the same schedules. To enable this nodes broadcast their own schedule. Before a node chooses its schedule, it first waits if it receives any schedules from neighboring nodes. A node only chooses a schedule on its own if it does not receive a schedule from any neighboring node; otherwise it chooses the received schedule. If a node receives more than one schedule, there are two ways to handle this. Either it adapts to both schedules or it stays with the schedule it received first. The disadvantage of the first case is less sleeping time since it has to be awake for both schedules. While the node in the second case only has to be awake for one schedule, there is another disadvantage. While the node can still talk to the nodes with the other schedule, if the node wants to broadcast packets to nodes from both broadcasts, it has to transmit the broadcast packet twice to enable all nodes to receive the packet.

To keep the nodes synchronized to be able to transmit messages to each other, the nodes have to transmit synchronization messages to counteract the different clock drifts of the nodes. The synchronization packets are pretty small and have to be transmitted periodically. The update period can be in the order of tens of seconds. In WSNs with very infrequent data transmissions this will still lead to a pretty high overhead, even though the synchronization packets are pretty small. The update packets also allow new nodes to adapt to the schedule of the other nodes, thus the initial listening period should be long enough to receive and adapt to an already existing schedule before choosing one itself. If S-MAC is used in combination with duty cycling it can happen that the off time of the duty cycle is longer than the synchronization interval of S-MAC. If this is the case then a time synchronization phase is necessary at the start of each duty cycle on phase.

One additional feature of S-MAC is the integrated collision and overhearing avoidance. For collision avoidance, the S-MAC implementation uses RTS (Ready To Send) and CTS (Clear To Send) packets. This helps to overcome the hidden station problem that occurs when two nodes that cannot hear each other try to transmit data to the same node at the same time. Furthermore, each packet in S-MAC also has a duration field. This field indicates the length of the remaining transmission. On the one hand this is used to indicate to other packets how long they have to wait until the transmission of the other node is finished and the node is able to request to transmit its own packets. On the other hand this information can also be used to implement overhearing avoidance. When a node receives either a RTS or a CTS message and it is not the intended receiver then it goes to sleep for the duration of the transmission.

B-MAC

B-MAC [54] which is the short for Berkeley-MAC has been developed at the University of California in Berkeley, as the name already implies. Unlike the S-MAC protocol that was presented before, B-MAC is an asynchronous low power listening protocol. To enable the reception of packets without using time synchronization, a preamble has to be sent before each data packet.

To guarantee that the intended receiver is awake when a data packet is transmitted, the preamble has to match the sleeping interval. This means that the preamble must be at least as long as the time between two awake phases of the receiver node. For the implementation of B-MAC in TinyOS for the Mica2 platform that uses the CC1000 radio controller, the preamble for a duty cycle of 11.5% has to be 250 bytes long. For a duty cycle of 2.22% the preamble even has to be 1212 bytes long [13]. The maximum length of the payload in TinyOS is only 29 bytes though. This means that the reduction of the idle listening time comes at the cost of an increased cost for transmitting packets. This also means that a lower duty cycle does not always lead to lower energy consumption, rather the duty cycle should be chosen according to the expected traffic profile of the network.

The B-MAC protocol does not provide an implementation for collision avoidance. Before transmitting a packet or when checking if there is a packet to receive on the air, B-MAC uses clear channel assessment (CCA) to check if the radio channel is currently free or if a transmission is currently happening. B-MAC provides the possibility to deactivate the CCA for the transmission part though. This allows the implementation of collision avoidance protocols like RTS\CTS on top of the B-MAC protocol. The nonexistent implementation of a collision avoidance protocol and an efficient implementation of the B-MAC protocol lead to a much smaller implementation size of B-MAC compared to S-MAC, leaving more of the limited memory of a sensor node to the implementation of the application itself.

One advantage of B-MAC is that the duty cycle of the implementation is not fixed. Not only can it be changed during runtime, but the B-MAC implementation provides an interface that allows the application that is running on top of B-MAC to set the duty cycle. Thus it is possible for the application to set the duty cycle to the value that is optimal for the traffic profile of the application and to change the duty cycle if the traffic profile should change.

UBMAC

Just like B-MAC, UBMAC [13] has also been developed at the University of California in Berkley. UBMAC is the abbreviation for uncertainty-driven BMAC. This protocol is a derivative of the standard B-MAC protocol. It is a hybrid protocol that estimates the clock drift between the nodes to be able to drastically reduce the required preamble.

The basis of UBMAC is the rate adaptive time synchronization (RATS) protocol that is proposed in [13]. In theory, the RATS protocol only needs to send a time synchronization packet about every 50 minutes to keep the time drift between two nodes below $90\mu s$ with a faulty rate below 5%. In the test implementation in TinyOS that was made for the Mica2 sensor board that is using the CC1000 radio interface, it was possible to keep two nodes within $225\mu s$ with sending a time synchronization packet about every 30 minutes and still keeping the faulty rate below 5%.

The estimated clock drift between the two nodes that want to communicate with each other is determining the length of the necessary preamble for packets sent with UBMAC. When both nodes are perfectly synchronized, the shortest possible preamble length that enables a successful communication is four bytes. The first two of those bytes are the minimum preamble size, the other two bytes are necessary to compensate for delays caused by software variations, by the radio on/off time and other unpredictable short delays. Each further byte added to the preamble increases the possible time difference between sender and receiver by $416\mu s$. The only packets that are sent with a preamble length that matches the sleep period are the time synchronization packets. This is necessary to guarantee that they arrive at their destination even if the drift between the nodes has suddenly changed.

UBMAC provides two different modes of operation that differ in the preamble. The first mode of operation is with a fixed preamble. To be able to keep the length of the preamble fixed, the RATS protocol has to keep the time uncertainty below the threshold that is determined by the byte length of the preamble. The second mode of operation uses a variable preamble. Here the RATS protocol just has to determine the time drift; UBMAC then sets the preamble length according to the time uncertainty. Which mode is suited better for a network depends on the amount of network traffic. For frequent data transmissions, the fixed preamble is better since the possible shorter preamble can outweigh the cost for the synchronization packets. With a variable preamble, the amount of synchronization packets can be reduced to one every few hours which will be better suited for networks with sparse network traffic.

Crankshaft

The Crankshaft MAC protocol [19] was specifically designed to deal with the problems of dense sensor networks. While most MAC protocols are designed for about five to ten neighbors, real-world deployments of WSNs can have fifteen or more neighbors. The high connectivity of dense networks leads to problems for MAC protocols that are either less severe or not even existent in sensor networks, where every node only has a few neighbors. These problems are overhearing, communication grouping, over-provisioning and the saving of the neighbor states.

Overhearing is already a problem in sparse sensor networks since every overheard packet is costing energy. In dense energy networks there are far more neighbors and thus also more packets that are overheard by a sensor node and more sensor nodes that overhear a packet. Communication grouping is used in some MAC protocols like S-MAC to separate communication frames into active and inactive parts. This allows the nodes to sleep in inactive parts of a frame but it increases contention and collisions in the network, especially if the sensor nodes have more neighbors then the amount that the MAC protocol was designed for. Over-provisioning occurs in MAC protocols that schedule sent-slots for all participating nodes. In dense networks each frame has to be split into many slots that will go to waste if a node has nothing to transmit. This increases latency because nodes have to wait for their transmit slots and wastes energy because nodes have to wake up at every slot and listen if the sender is using its slot and if the data is for them. Memory space is limited on sensor nodes, thus MAC protocols that have to save a state for each of their neighbors can lead to problems in dense networks. Reducing the needed memory by discarding the states of some of the neighbors also leads to problems since it can hinder communication and cause problems with the routing layer that keeps its own list of neighbors.

The basic principle of the Crankshaft MAC protocol is that instead of dividing a frame into sender slots, a frame is divided into receiver slots. The receiving slot of a node always has the same offset from the frame start. Since nodes only listen in their own receiving slot and those are different between nodes this greatly reduces overhearing. To keep the size of the frames low, a frame has a limited amount n of unicast slots. Since there are more than n nodes in a dense network, there is not a unique slot for each node of the network. The nodes have to share slots, which slot a node is using is determined by its MAC address modulo n. Each slot starts with a congestion window to decide which node is allowed to transit if more than one node wants to send data to the current receiving node. When the receiving node wakes up the node that won the contention for the slot starts transmitting its data. Since the unicast slots are not suited for broadcasting each frame also has some broadcasting slots where all nodes are awake for receiving. Since each node needs to know exactly when a frame starts Crankshaft requires time synchronization amongst the sensor nodes which could be achieved through a reference node for example.

2.3.3 Balancing Residual Energy Amongst Sensor Nodes

When a node gets completely turned off during duty cycling which includes turning off the power supply for the volatile memory of the node, all temporary data of the node needs to be stored in non-volatile memory of the node and then be read out again when the data is needed. When using NC in a network, the nodes in the network will often have to wait for packets from other nodes in the network before they can combine the packets and transmit them again. If the node gets duty cycled between the reception of those two packets the packet that is received first has to be stored in the non-volatile memory and read out again afterwards which leads to an increased power consumption.

In [14] an energy management approach is proposed that allows trading in additional transmission on one node to avoid having to store packets in the non volatile storage on another node. Instead of storing a packet in the non-volatile memory it is also possible that the sender of the packet just retransmits the packet again in the duty cycle period during which the node also receives the other packet so that the node can immediately send out the combined packet. While this might seem counterproductive since the total amount of energy that is needed in the network may be increased there is one big advantage of this approach. When the sensor network is powered by an EHD the amount of energy that is available to the nodes in the network will differ. The proposed energy management allows reducing the energy consumption on nodes with a worse energy harvesting profile by increasing the energy consumption on nodes in the network with a higher amount of harvested energy thus allowing the implementation of NC to adapt its power consumption in the network to the harvesting profile of the network.

2.4 Existing Middleware Implementations

The development of applications for WSNs can be quite time consuming. One reason for this is that standard application parts like networking, data aggregation, data fusion or power management have to be implemented again for every new application. Even hardware interaction can be part of the software development if the application is not built upon a simple operating system for WSNs such as TinyOS. The purpose of a middleware is to operate between the sensor node hardware or the operating system and the application and provide standard applications to the software running on top.

The already existing middleware do not offer the same amount of functionality though, they differ in the type and amount of services that are provided. An overview on the challenges that have to be addressed by middleware for WSNs are presented in [17][18]. Sensor nodes are usually limited in available energy, memory, bandwidth and computational power. A middleware should provide ways for the efficient use of the resources while using the available resources efficiently itself. Another important part of a middleware is network communication. There are several requirements for the routing implementation of a middleware. The middleware should offer good scalability and offer a robust network communication that is able to cope with problems like device failures, interferences and packet loss. Since not all nodes of a WSN have to be of the same type, a middleware should be able to deal with heterogeneous sensor nodes and provide a hardware independent interface to the applications running on top of the middleware. Further requirements on a middleware include handling of interactions with real world applications, providing means of data aggregation, offering the mechanisms for QoS, provide some means for an application to gain knowledge about network parameters and offer some security features.

Middleware can be classified into five main categories [17][18]. The first category of middleware is virtual machines. This type of middleware lets the user write their applications in separate, small modules that can be uploaded onto the sensor nodes over the network. The virtual machine on the sensor node then interprets the module and executes the application. The downside of this approach is the overhead that is introduced by the virtual machine. The second category is modular programming also known as mobile agents. The idea behind this approach is to make applications as modular as possible.

This allows an easy injection and distribution of new code without the need to replace the whole application. The update with modular programming consumes far less energy than always replacing whole applications since the modules are much smaller than whole applications. The third category of middleware is based on a database approach. Here the whole network is considered as a virtual database. This provides an easy interface to the user that allows data acquisition over the network by using queries just like in normal databases. The disadvantages are that only approximate results are provided and that there is no possibility for the detection of spatio-temporal relationships between events. The fourth category is application driven middleware. The main difference to other middleware approaches is that application driven middleware allows the application running on top of the middleware to interact with the network layer. The disadvantage of letting the user control the network operations management is that this requires a tight coupling between the middleware and the applications which might result in specialized middleware that is not suited for other applications. The last category is message-oriented middleware. This approach is based on the publish-subscribe mechanism that simplifies the message exchange between source and sink nodes. It also naturally supports asynchronous communication that allows a loose coupling between sender and receiver nodes which is well suited for event based applications like they are common in WSNs. The rest of this section presents the most popular middleware approaches of the presented categories with a special focus on implemented power management strategies.

2.4.1 Maté

The Maté middleware [37] is developed at the University of California in Berkeley and is a representative of the virtual machine category. Maté is a byte code interpreter that is running on top of TinyOS on the wireless sensor nodes. It is built as a single component in TinyOS that sits on top of several system components. These components include the sensors, the network stack and the non-volatile storage. Maté is mainly aimed at sensor networks that require frequent reprogramming of the sensor nodes. The concept of Maté is to lessen the constraints by the limited bandwidth and the high cost of communication for application updates by offering a flexible reprogrammability of the applications that are running on the virtual machine. Maté has a built-in routing algorithm but it is possible to implement own routing algorithms on top of the virtual machine. One interesting concept of Maté is that it hides all asynchronicity from the user applications. For example it will suspend an application when sending a message until the sending event is completed. This simplifies application development on the user side and makes it less prune to bugs.

The code that is running on top of the virtual machine in Maté is broken up into small capsules. Each capsule consists of up to 24 instructions; larger applications can be composed of multiple capsules. The reason for 24 instructions is that this allows a capsule to fit into a single packet in TinyOS. For updating, each capsule of an application can be updated independently. This allows updating parts of an application without having to transmit the whole application again thus keeping the overhead small.

The downside of the virtual machine approach is that Maté adds some computational overhead to the execution of an application since Maté has to execute additional instructions for the interpretation of the byte code. The computational overhead can be compensated though by the reduced cost for application updating when frequent updates of the running application are necessary. Maté does not provide any power management strategies to the user applications, though it might be possible to realize power management on the user side.

2.4.2 Impala

The Impala middleware [43] was designed as part of the ZebraNet project [26], a wildlife tracking project, but can also be applied to other types of sensor networks. The Impala middleware belongs to the category of modular programming. The system architecture of the middleware is separated into two layers. The upper layer contains the applications; only one application can run at a time. The lower layer contains the three middleware agents called application adapter, application updater and event filter. The application adapter is responsible for adapting the applications to certain runtime conditions to enable better performance, energy efficiency or robustness. The application updater handles the reception and propagation of software updates over the radio module and the installation of the updates. The event filter is responsible for the dispatching and processing of events from the hardware. Processing is done sequentially to reduce the programming complexity for the applications.

Power management in Impala is handled by the application adapter. The sensor nodes can be loaded with multiple applications to handle the same task, for example multiple routing algorithms that are preferable under different conditions of the sensor node and the network. The runtime states of a sensor node are represented by application parameters of a specific application and system parameters of Impala itself. Application parameters for a routing algorithm can for example be the number of neighbors or the amount of data that was successfully transmitted, system parameters could be the battery level or the transmitter power. The decisions of the application adapter are based on these parameters, like changing to a routing algorithm with lower energy consumption if the battery level drops under a certain threshold. One downside of the application adapter is that it is currently only based on local parameters and does not support adaption based on global parameters of the network. The application adapter also allows reacting to device failures in the same way by changing to an application that does not need the device, if possible.

2.4.3 SINA

The SINA middleware [63] was developed at the University of Delaware, the abbreviation stands for Sensor Information Network Architecture. SINA belongs to the category of database based middleware, modeling the sensor network as a distributed database. The sensor network in SINA can be seen as a collection of datasheets where each sensor node is representing a datasheet. A datasheet itself contains a collection of attributes, also called cells that represent the state of the sensor node and its sensors. The architecture of SINA relies on three concepts for its operation. The first concept is hierarchical clustering that aggregates sensor nodes in clusters according to location and power levels of the sensor nodes. A cluster head is elected in each cluster that is responsible for information filtering, fusion and aggregation. The second concept is attribute-based naming. To increase the scalability, SINA does not address sensor nodes directly but rather addresses nodes by properties like their location or a sensor value. The third concept is location awareness which is needed to enable regional data queries.

The query mechanic of the data gathering in SINA hides the actual network communication from the user, when submitting a query it is not required to define how the information will be collected inside the network. The only power management that is offered by SINA is hierarchical clustering. This allows to reduce energy consumption since the cluster head can collect data and only the combined data has to be sent to the base station thus reducing the necessary transmissions. While the cluster head can be chosen according to the energy level of the sensor nodes, SINA also supports reinitiating of the clustering if a cluster head runs low on power or fails completely. This can lead to a fair spreading of the additional energy cost of being the cluster head according to the energy that is available to the sensor nodes in the network.

2.4.4 MiLAN

The MiLAN middleware [21] is developed at the University of Manchester and belongs to the category of application driven middleware. The abbreviation MiLAN stands for Middleware Linking Applications and Networks. The MiLAN middleware uses the description of the application requirements that it receives and the information that it can gather about the network conditions to optimize the sensor and network configurations of the whole network. Unlike other middleware, MiLAN is not only situated between the operating system and the application layer, instead the architecture of MiLAN is extended into the network protocol stack. To enable the support of different physical networks, MiLAN provides a high-level abstraction layer that converts the middleware commands to the protocol specific commands for the used network type.

Power management in MiLAN is handled by adapting the configurations of the sensors and the network. The general goal here is to configure the sensors and the network to optimize the application lifetime while still meeting the QoS that is requested by the application. This means intelligently choosing how long each feasible set of sensors is used. To further increase the lifetime of the application MiLAN offers the possibility to trade off QoS for an increased lifetime. By setting the QoS requirements for an application MiLAN allows the user to have influence on the power management of the sensor network in a relative simple way without having to deal with in-depth knowledge on power management algorithms.

2.4.5 Mires

Mires [64] is an example of a message-oriented middleware that is running on top of TinyOS and is implementing a publish/subscribe communication for WSNs. The Mires middleware consists of several services, namely the publish/subscribe service, the routing service and possible additional services. The general idea of publish/subscribe is that information is published by suppliers and forwarded to the one or more subscribers. To make it possible to only subscribe to the data of a supplier that is of interest for a certain subscriber, the published information can be associated to a certain topic that can be subscribed individually. Informing the subscribers of a new message is done by the notification service that is buffering the messages in the meanwhile; this allows an asynchronous communication between data producers and consumers. The flow of publish/subscribe consists of three phases. In the first phase the nodes in the network have to advertise their available topics. In the next phase the advertising messages have to be routed to the sink nodes to allow them to choose the topics that they want to subscribe. In the last phase the sink nodes then have to broadcast their subscribed messages to inform the nodes which topics they have to publish to which nodes.

The Mires middleware does not provide any power management strategies but it offers some ways for the user to influence the energy consumption. The first way of influence is the routing algorithm. While Mires offers a built in routing algorithm it also offers the possibility to use a different routing algorithm as long as it implements the interface that is required by Mires. The second way of influence are the possible additional services. They can be easily incorporated into Mires through interfaces that define notification events. This allows the user to add services like data aggregation to existing middleware.

2.5 Summary

This chapter presented related work on different topics that each have in common that they deal with WSNs. While there already exists a lot of work on the general topics that were presented in this chapter, the situation changes when looking at related work that specifically deals with the problems of energy harvesting powered WSNs. While there is some work on routing and duty cycling that specifically deals with the problems of energy harvesting powered WSNs, the section on existing middleware implementations reveals that there is not yet any middleware that is specifically designed for this problem setting. In this thesis a networking framework is proposed that aims at combining and adapting existing work on energy harvesting powered WSNs to release designers of an application for an energy harvesting powered WSN from having to deal with routing and energy management. The proposed networking framework also implements NC as a possibility to reduce the amount of necessary data transmissions in a network in comparison to simple data routing. ONC being a new topic of research for 802.11 networks to increase the throughput of networks, this framework will port ONC for WSNs to explore its possibilities to reduce the energy consumption of data routing.

Chapter 3

Design

The last chapter gave on overview on already existing approaches for several technologies that are of interest for energy harvesting WSNs. While many of the ideas presented in the last chapter show great potential for the use in energy harvesting WSNs there is not yet a framework that provides a combined implementation of the presented technologies. This chapter presents the design of a networking framework for energy harvesting WSNs.

Each of the following sections focuses on a different aspect of the design. The first section presents the basic design of the implementation, a short overview of the parts required for the networking framework and the constraints of the proposed framework. The second section focuses on the design of the chosen EHAR algorithm. The third section proposes a design for the use of NC in energy harvesting WSNs and the possibility of using ONC on top of EHAR. The next two sections present the design of a duty cycling scheme and additional low power listening that operates on top of the duty cycling algorithm. The last section presents the tests that are done to evaluate the performance of the framework.

3.1 Basic Design

To be able to design the different parts necessary for the proposed networking framework, the problem settings and the goals of the framework need to be defined first. This section presents the overall design and defines the problems that have to be addressed by the framework as well as the goals that the framework aims to solve according to its overall design. These considerations are then used to select the necessary parts that are presented in further detail in the following sections.

The network setup that is taken as a basis for the framework is a WSN with an energy harvesting device as power supply on each node of the network. The networking framework is not designed for a random deployment of the sensor nodes as the underlying structure of the network, but the structure of the network is a grid of sensor nodes as can be seen in Figure 3.1. The network consists of a grid of $n \times n$ sensor nodes, where each node can communicate directly with all of its up to 8 neighboring sensor nodes. The network is not designed as a classical WSN where every node is sensing data and sending that data to one base station. Rather, the nodes on the top and on the left of the network are sensing data and the nodes on the bottom and on the right of the network are receiving the sensed data. An exception are the sensor nodes on the edges of the network, which are neither sensing nor receiving data but just participate in data routing. They are not actually needed; the networking framework is also able to perform correctly without these nodes. Each sensing node has a corresponding destination node that it is sending its data to. The node A_i on the left side of the network is transmitting its sensed data to the node C_i on the right side of the network for example. So the nodes $A_2, ..., A_i, ..., A_{n-1}$ on the left side of the network transmit data to the corresponding nodes $C_2, ..., C_i, ..., C_{n-1}$ on the right side of the network and the nodes $B_2, ..., B_i, ..., B_{n-1}$ on the top of the network transmit data to the corresponding nodes $D_2, ..., D_i, ..., D_{n-1}$ on the bottom of the network. For testing purposes the network configuration can also be adapted to a single data sink network. In this case, node D_{n-1} acts as data sink and all other nodes in the network are data sources that transmit their data to node D_{n-1}.

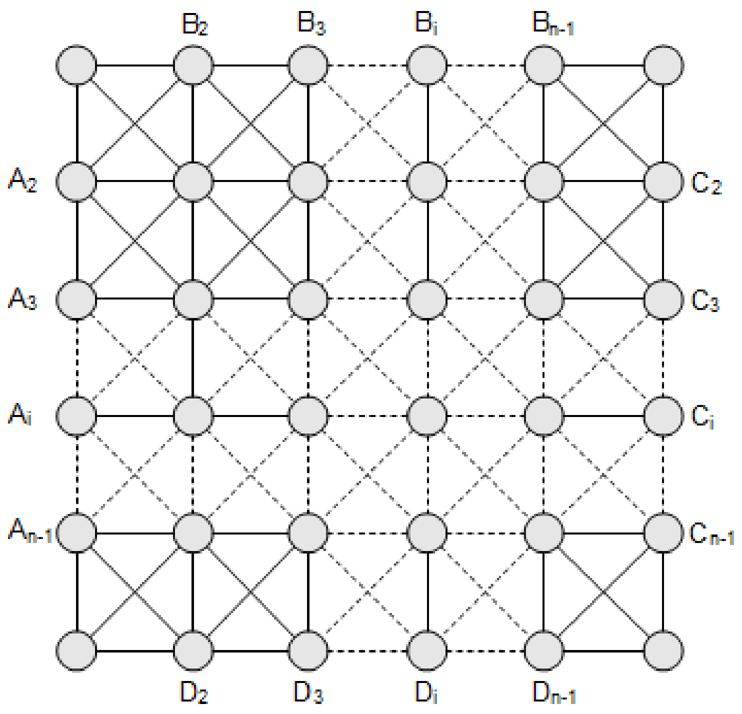


Figure 3.1: Network configuration for $n \times n$ nodes

3.1.1 Problems

While the problems for networking of energy harvesting WSNs have similarities to the problems of other wireless networks like 802.11 and the problems of battery powered WSNs, they are not the same. This section presents the problems that need to be solved by a networking framework for energy harvesting WSNs but also looks at challenges that are not addressed by the framework.

As can be seen from the network setup in Figure 3.1 the network consists of several data sources and data sinks. This also means that each node in the network needs to keep routing information for several data sinks. But each node only keeps routing information for the nodes of the network that act as data sinks and all data destined for other nodes of the network has to be transmitted via broadcasting. The topology also defines that each node in the network can only transmit data directly to its neighbors. This means that packets from data sources to data sinks have to be transmitted via the nodes that lie between thus making it necessary that the network framework supports multi hop routing. Since the network setup does not define the amount of sensor nodes in the network, the networking framework has to be able to work both in small and large versions of the presented network setup. Since in large setups most of the nodes in the network will have 8 neighbors that all use the radio interface, packet collisions will be a big problem for the networking framework. Therefore the networking framework has to implement some sort of collision avoidance mechanism. Furthermore, the routing algorithms that are implemented in the framework need to be fault tolerant to transmission errors since even without packet collisions transmission errors will occur.

All sensor nodes of the network are of the same type, there are no nodes with a better EHD, larger energy storage or a faster processor. This means that there are no nodes that could effectively take the role of a cluster head without running out of energy much faster than the rest of the network. Moreover, since the transmission range of the sensor nodes is rather limited, a large portion of the network would have to be cluster heads or at least still participate in routing. Thus the networking framework does not use hierarchical-based routing.

All nodes in the network are powered with an EHD as their power supply. While the EHD of each node has limited energy storage, the main power for operating the node is the harvested power. The energy storage is only meant to be able to deal with fluctuations in the harvested energy and not as main power supply that is supported by the harvesting device to increase the node survival time. Since the energy storage of a node is limited, the networking framework has to be able to adapt its power consumption to the amount of harvested energy to prevent the node from running out of power. Due to the use of energy harvesting the available amount of power on each node of the network is different. The differences in the provided power from the EHD between the nodes of the network are caused to a small extent by the differences of component properties, but to a much larger extent by properties of the used energy harvesting source. For an EHD with a solar panel for example, the harvested energy will greatly differ whether a node is in the sun or in the shade or whether the sun is hidden behind clouds or not. Specially in large networks the terms for energy harvesting will never be the same for all nodes in the network. The networking framework needs to balance network traffic according to the energy level of all nodes of the network and be able to adapt to changing energy level distributions in the network.

Another problem for the networking framework is the energy consumption of the nodes while they are in idle mode. Even if the modules of a node have nothing to do at the moment they still consume power. For sensor nodes with a very constrained amount of available power, as is the case for nodes powered by energy harvesting, the idle power consumption can cause that nodes no longer have enough power available to fulfill their actual tasks. The radio module for example will still check the radio interface for packets all the time even if there are no incoming transmissions in the near future, which leads to a huge idle power consumption.

There are also some aspects that are not addressed by the networking framework, for example mobility, quality of service and latency. While sensor nodes are mobile in several already implemented WSNs, this is not the case for all WSNs. For the proposed setup, the networking framework assumes that all nodes are stationary. QoS and latency are usually very important for 802.11 networks, but of lesser importance for most WSNs. The focus of the networking framework lies in an energy efficient communication specifically addressing the problems arising by an energy harvesting power supply. The energy efficient communication of the networking framework accepts an increased latency and a worse QoS in order to handle other problems of the framework more efficiently.

3.1.2 Goals

While the last section focused on the problems that have to be addressed by the networking framework, this section presents the goals that should be achieved by the networking framework. And it also takes a look at common goals that are not in the focus of the networking framework as well as the reasons for that.

The biggest goal for the networking framework is to achieve energy neutral operation within the whole network. Due to limited energy storage on the nodes, designed to only compensate for fluctuations in the harvested energy but not to power a sensor node over a longer period of time, the possible power consumption of the nodes strongly depends on the power provided by the EHD. This means that the node has to be able to adapt its power consumption to changes in the provided energy. The goal for the networking framework is not only to enable a long lifetime of the total network, but also to keep all sensor nodes in the network alive infinitely as long as there are no technical defects.

The differences in the amount of harvested power between different nodes in the network leads to a different energy potential of the nodes. The networking frameworks has to balance network traffic according to the pattern of the energy harvesting potential of the different nodes in the network. Since the differences in harvested energy between the nodes change over time, the networking framework also has to be able to change the routing distribution to adapt to the changing harvesting profiles. The goal for the networking framework is to balance the network traffic amongst the sensor nodes as good as possible to prevent any node of the network from having to route more packets than it can sustain with its harvested energy.

But even if the traffic is spread optimally over the network according to the harvested energy, each transmitted packet still consumes a reasonable amount of available power. Another goal for the networking framework thus is to keep the amount of packet transmissions as low as possible. The networking framework has no influence on the amount of data packets sent out by the source nodes since this is controlled by the user applications that are using the framework. But even if the networking framework has no influence on the amount of data packets that are sent out by the data sources the framework should aim at reducing the necessary transmissions to send all data to its intended data sink. The networking framework can also influence the amount of transmissions that are necessary to distribute control information in the network.

Another goal of the networking framework is to reduce the idle power consumption of the different modules of a sensor node. Every module of a sensor node still consumes power even if it has nothing to do, with power consumption depending on the module. For example, an idle temperature sensor consumes far less power than an idle radio module that constantly checks the radio interface for incoming packet transmissions. The networking framework has to implement a method to reduce the idle power consumption of the different modules to lower overall idle power consumption. The radio module poses a special challenge here since all nodes still need to be able to communicate with the other nodes in the network. But since the radio module is also amongst the modules with the highest idle power consumption the benefits outweigh the increase in complexity for overcoming these challenges.

To be able to achieve the aforementioned goals the networking framework has to deal with some restrictions for other features of the framework. Due to the limited amount of available energy the achievable data rates of the framework will be rather low. Optimizing the data rates would lead to a great increase in energy consumption and would not permit an energy neutral operation of the network. While spreading the network traffic is good for balancing the energy consumption required by data routing it might increase the latency of the data packets. Optimizing the networking framework for low latency would reduce the flexibility of the framework and might overuse single nodes that offer good possibilities for low latency values. A tradeoff has to be made here between flexibility and latency. While the main goal is to allow flexible data transmissions for a better spread of the network traffic and a lower energy consumption of the nodes, the latency increase should still not be completely ignored, otherwise transmitted data might no longer be of any use when it finally arrives at its destination.

3.1.3 Parts of the Networking Framework

The last two sections presented the problems that have to be solved by the networking framework and the goals to be achieved. These problems and goals are the basis for selecting the algorithms of the networking framework and for their detailed design. This section presents the algorithms that are part of the networking framework and the reasons for choosing them. The following sections will then go into further details on the specific algorithms.

To solve the problem of spreading the network traffic and enabling a balanced communication in the network according to the energy harvesting profile of the network the network framework uses EHAR. The used EHAR protocol will apply probabilistic forwarding with probabilities that are related to the power delivered by the EHD of the nodes for spreading the traffic according to the energy harvesting profile of the network.

The networking framework will also implement NC to be able to combine packet transmissions and reduce the total amount of needed transmissions. Two variants of NC will be provided by the networking framework. The first variant is normal NC that will replace EHAR if it is activated. NC will allow to considerably reduce the amount of necessary transmission to get all data to its destination, but the flexibility of EHAR is then lost. The second variant is ONC which is operating on top of the used EHAR protocol. The goal for ONC is to detect possibilities to send out combined data packets instead of single packets during normal routing to save some data transmissions.

To overcome the problem of power consumption in idle mode that is draining the available power and preventing energy neutral operation of the network, the framework will implement duty cycling. By periodically turning off the different modules of the sensor nodes when they have nothing to do the power drain of idle modules can be greatly reduced. To further reduce the idle power consumption of the rather power hungry radio module the networking framework will also implement a low power listening scheme on top of the duty cycling of the radio module.

The problem of transmission errors can be divided into two main categories, packet collisions and lost packets. To prevent the occurrence of packet collisions on the radio interface the networking framework will implement a time slotted transmission scheme for the nodes in the network. This means that in each time slot only one sensor node is allowed to send, which removes the possibility of packet collisions if the nodes in the network are correctly time synchronized. The time frames will be chosen slightly larger than needed to remove the need for very accurate time synchronization which would lead to very high overhead to remain accurate. The detection of lost packets is a bit more complicated, since most packets are sent in broadcasting mode where packet acknowledging is not supported. The networking framework will provide possibilities for NC and ONC to detect if they miss a packet that they need for decoding and will allow to request the retransmission of the missing packets.

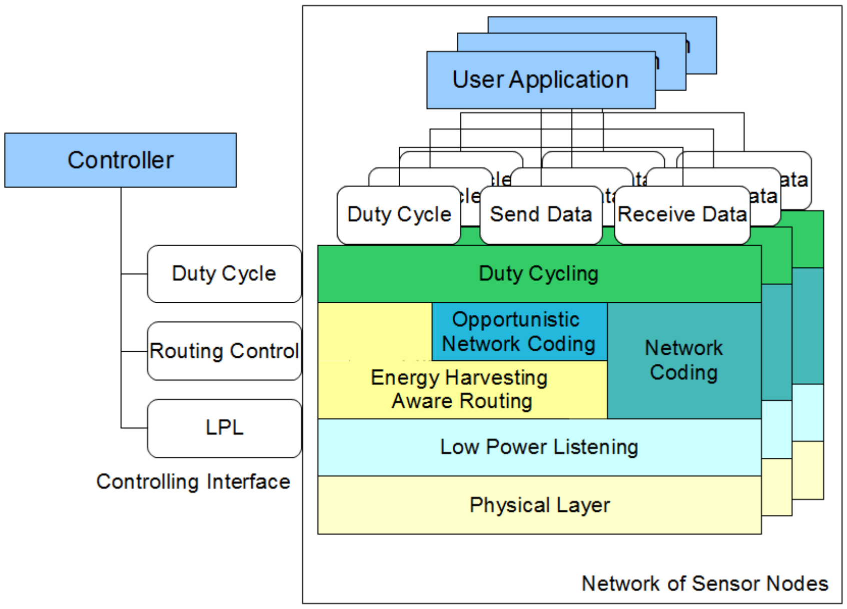


Figure 3.2: Layer model of the networking framework

Figure 3.2 presents the layer model of the proposed networking framework. The lowest layer is the physical layer. In the MAC layer that lies above the physical layer, low power listening is implemented. The network layer consists of two independent implementations, NC and EHAR. ONC is built upon EHAR since it does not change the routing itself but just the content of the actually transmitted packets. Duty Cycling is situated on top of the implemented routing algorithms. Furthermore, the networking framework also provides interfaces to a user application that is running on top of the framework and a controller outside the network. The interfaces that are provided to a user application are send data, receive data and a duty cycling interface. The controller node can influence duty cycling, low power listening and the routing algorithms of the networking framework via the provided interfaces.

The networking framework also provides an interface for communication with a base station outside of the basic network structure. The interface provides mechanisms to control the functionality of the framework and change parameters during runtime. The communication with the base station is done by one of the nodes of the network that operates as a controller node. This node is responsible for gathering data needed data from the network and for starting the distribution of commands and parameter changes from the base station over the network. This allows to reconfigure some aspects of the networking framework without having to collect all nodes of the network, reprogram them and then deploy them again.

3.2 Energy Harvesting Aware Routing

The routing of data packets in the networking framework is handled by an EHAR algorithm. This section presents the design for the chosen algorithm, based on the approaches for EHAR presented in section 2.1.3.

The chosen algorithm is based on the EHAR algorithm by Kansal et al. that was proposed in [28] [29]. They propose a probabilistic routing scheme to balance the energy cost of transmitting over different nodes in the network. The probabilities of the possible next hops for a packet are based on the energy potentials of the next hop nodes. The energy potential of a node depends on the harvesting rate of the EHD and the energy level of the energy storage. But for routing packets first the routing information and the energy potential of the nodes has to be distributed over the network.

To be able to route data from its source to its destination, every node on the routing path needs to have routing information for the destination stored in its routing table. This means that since each node in the network has to be capable to forward data to every destination node, each node in the network also has to store routing information for every destination node in its routing table. Therefore each node has to store routing information for the destination nodes $C_2, ..., C_i, ..., C_{n-1}$ and $D_2, ..., D_i, ..., D_{n-1}$ from Figure 3.1. Since the used routing scheme is a probabilistic routing scheme every node has to store the routing information for all possible next hops that can route packets to the destination within a certain threshold of the costs of the optimal path.

To be able to find the optimal path and set the threshold for the deviations from the optimal path the criteria for the optimal path has to be defined first. For the networking framework the optimality criteria for routing will be the hop count. So the optimal path between a node and a destination node is always the path with the lowest hop count which equals to the fewest needed transmissions to send a packet from the current node to the destination node. For EHAR a node will not only save routing information for the path that offers the lowest hop counts though. To enable probabilistic routing a node needs to have multiple entries for each destination. Each sensor node stores routing information for all paths that either have a hop count equal to the optimal path or not larger than the optimal hop count plus a fixed threshold value.

Before being able to route data, each node needs to build up its routing table. The distribution of the routing information is done by broadcasting and needs to be done separately for each destination node. For constructing the routing tables each node needs a defined set of information about the possible routes to the destinations. The routing information packets that are used to exchange routing information thus consist of the destination ID, the next hop ID, the hop count, the energy potential and a sequence number. While the destination ID does not change when a node receives a routing information packet and broadcasts it again, all other values change for each new broadcast. The next hop ID is set to the ID of the current node that is broadcasting the packet, the hop count is simply increased by one and the energy potential is set to the energy potential of the current node. The value of the energy potential is determined according to equation 2.7. The sequence number is used to determine if a routing information packet is fresh and is only increased by the destination node when it starts broadcasting its routing information again to enforce a routing table update.

The exchanging of routing information is started by the destination that wants to distribute its routing information or update the routing information that the nodes have already saved. To inform the other nodes that there is a destination, a routing information packet is broadcasted where the destination ID is set to the ID of the current node. Since the current node is also the destination node, the value of the next hop ID is equal to the destination ID value. The hop count is initialized to one and the energy potential is set to the energy potential of the destination node. If this is the first distribution of routing information from this destination node then the sequence number is initialized to an arbitrary value, if the routing information is sent out to update routing tables then the new sequence number is set to a value higher than the sequence number used last.

When a node receives a routing information packet, it first checks its routing table whether there is already an entry for the destination ID of the routing information packet or not. If the node does not yet have an entry in its routing table for that destination ID then a new entry is created. Since this is the first entry for that destination ID the hop count of the received routing information will now also be the new minimum value for the hop count to this destination. After saving the routing information in its routing table the node needs to update the routing information packet and then broadcast it again. The next hop ID is set to the ID of the current node, the hop count is increased by one and the energy potential value is set to the energy potential of the current node, all other values of the packet stay the same. This updated routing information packet is then broadcasted out again to spread the information over the network.

If a node already has an entry in its routing table for a destination node, further checks need to be done to decide whether the received routing information should be stored in the routing table or not. If the node already has an entry for the same destination ID and the same next hop ID, the node compares the hop count of the new routing information to the already stored hop count value. If the new hop count is lower than the one already stored, the stored routing information is overwritten with the new routing information, else the new routing information is discarded. If there is no entry yet in the routing table for the next hop ID of the received routing information then the node has to decide if the new information is worth storing or not. Thus the node compares the hop count of the received routing information with the minimum hop count of already stored routing information for the destination ID. If the hop count of the new routing information is within a fixed threshold of the minimum hop count then the information is stored in the routing table, else it is discarded. Since the routing table already contained an entry for the destination ID the node only broadcasts the routing information packet again if the hop count of the received routing information is below the old minimum hop count of the node.

The value of the threshold has a major influence on the amount of possible different routes from a node to a destination node. To keep the detours of a packet on its way to its destination within limits, the threshold for the hop count for the networking framework will be set to one. This still allows to route packets around areas with low energy while it prevents that a packet can actually take longer detours in the wrong direction on its way to its destination. To further limit the detours of a packet on its way to its destination, the amount of times a packet is allowed to be routed over next hops above the minimum hop count is limited. Each packet is only allowed to take one additional hop on its way to the destination. For data transmissions from node A_2 to node C_2 in Figure 3.1 without any additional hops there are only 18 different paths. With the one additional hop that a packet is allowed to take somewhere along the route to its destination, the amount of possible routes increases to 108.

When the sequence number of a received routing information packet is higher than the previous sequence number, the stored routing information is outdated. Since there is no way for a node to know whether the stored information is still valid, the old routing information for that destination ID has to be dropped. This holds true even if the hop count of the new routing information is above the value of the stored minimum hop count plus the threshold, because the node does not know if the route with the old minimum hop count is still valid. The received routing information is then entered as the first entry for the destination ID in the routing table. After that the received routing information packet is updated and broadcasted again. From now on all packets with the new sequence number are treated as the active routing information packets by the current node, all routing information packets with a lower sequence number are immediately dropped.

As soon as a node has at least one entry for a destination ID it can forward packets to that destination node. But to be able to take advantage of the probabilistic property of the routing algorithm, a node needs more than one entry for a destination ID in its routing table. The probabilities of the different paths to be chosen for the next hop depend on the energy potential of the next hop. In Figure 3.3 node A wants to transmit a packet to node B. Since it cannot send the packet directly, it has to forward it via one of the nodes C, D and E.

As can be seen in Figure 3.3 the three possible nodes for relaying packets all have different values for the harvesting rate of the EHD and the power that is currently stored in the battery. This means that they will also have different values for the energy potential. As can be seen in equation 3.1, the energy potential E_i of node i not only depends on the harvesting rate of the EHD ρ_i and the residual battery level B_i but also on a weighting factor ω.

$$E_i = \omega * \rho_i + (1 - \omega) * B_i \tag{3.1}$$

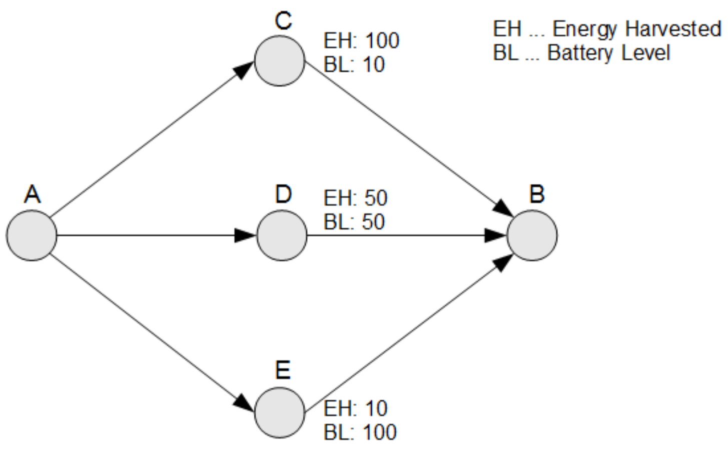


Figure 3.3: Example setup for EHAR

To analyze the influence of the weighting factor ω on the energy potential of a node, three cases will be compared. For the first case, ω is set to 0.9 which means that the harvesting rate of the EHD will have the main influence on the energy potential of the node. For node C this leads to a value for the energy potential E_C of:

$$E_C = 0.9 * 90 + 0.1 * 10 = 82 \tag{3.2}$$

Inserting the numbers for the nodes D and E into equation 3.1 results in an energy potential E_D for node D of 50 and an energy potential E_E for node E of 18. Here it can clearly be seen that the node with the highest harvesting rate of the EHD also has the highest energy potential. When the weighting factor ω is changed to 0.5 the energy potential for all three nodes C, D and E is the same $E_C = E_D = E_E = 50$. Since the sum of the values for the harvesting rate of the EHD and the residual battery level are the same for all three nodes the energy potential is also the same for all three nodes with a weighting factor ω of 0.5. When the weighting factor is lowered to 0.1 the residual battery level becomes the main influence of the energy potential of the nodes. The energy potential E_C of node C is lowered to 18 while the energy potential E_D of node D stays constant at 50. But the node with the highest energy potential now is node E with an energy potential value E_E of 82.

One of the main goals of the networking framework is energy neutral operation. For the routing protocol this means that it is favorable to transmit packets via nodes that have an EHD with a high energy harvesting rate since they can recover faster from the power that they have to spend on transmitting a packet. This means for the selection of the weighting factor ω for the calculation of the energy potential for the EHAR protocol that a value close to one is preferential. With a value close to one, the nodes with a high energy harvesting rate will have a higher energy potential than nodes with a low harvesting rate even if they have a worse residual battery level.

To calculate the probabilities of the three different next hops for transmitting the packet at node A, the edge costs of the three connections have to be calculated first. Since

Node	Energy Potential	Edge Cost	Probability
C	82	$0.0\overline{12195}$	0.543
D	50	0.02	0.331
E	19	$0.0\overline{5}$	0.126

Table 3.1: Example values for EHAR in Figure 3.3

a weighting factor ω close to one is preferential for the problem setting of the networking framework the values for ω will be set to 0.9 for the networking framework. As defined in equation 2.8 the edge cost of a connection between two nodes is the inverse of the energy potential of the receiver node. For the edge e_{AC} that represents the communication link between the sender node A and the receiver node C the edge cost $c_{AC}(e_{AC})$ can be calculated as:

$$c_{AC}(e_{AC}) = 1/82 = 0.0\overline{12195} \tag{3.3}$$

The edge cost $c_{AD}(e_{AD})$ for the receiver node D amounts to 0.02 and the edge cost $c_{AE}(e_{AE})$ for the receiver node E amounts to $0.0\overline{5}$.

With the edge costs of the three possible connections, node A is now able to calculate the probabilities for choosing node C, D or E for the next hop. The probabilities depend both on the edge cost of the used connection and on the sum of the cost of all edges. Inserting the values for the edge e_{AC} into equation 2.9 gives the following probability P_{AC} for the connection from node A to node C:

$$P_{AC} = \frac{\frac{1}{0.0\overline{12195}}}{\frac{1}{0.0\overline{12195}} + \frac{1}{0.02} + \frac{1}{0.0\overline{5}}} = \frac{82}{151} = 0.543 \tag{3.4}$$

Inserting the values for the edge e_{AD} between the nodes A and D gives a probability P_{AD} of 0.331 and the probability P_{AE} for the edge e_{AE} between the nodes A and E is 0.126. This means that more than half of the packets will be routed via node C and only about an eighth of the packets are routed via node E, the node with the lowest energy harvesting rate. Table 3.1 gives an overview on the values for the energy potentials of the different nodes as well as the edge costs and probabilities of the different connections.

3.3 Network Coding

Since data transmissions are very expensive regarding energy, the networking framework uses NC to reduce the amount of messages that need to be transmitted. The networking framework implements two different versions of NC. The first version is normal NC that replaces EHAR for data transmission. The encoding of the packets is not only done for single hop transmissions, but the encoding scheme is based on the routes of all data flows in the network. The design is presented in detail in the following paragraphs. The second version is ONC which is operating on top of the implemented EHAR algorithm. The goal is to find opportunities for NC during routing to reduce the necessary amount of transmissions to forward all packets on a node. The detailed design is presented in the following subsection.

When all sensing nodes are sensing and sending data to its destination, each of the intermediate nodes in the network has to transmit two packets per sensing cycle if the traffic is spread evenly across the network. This means that all nodes in the middle have to send twice the amount of packets that are sent out by the sensing nodes. When looking at the bandwidth this means that when the relaying nodes are completely utilizing their available bandwidth the sensing nodes are only able to utilize half of their available bandwidth. The same happens when we assume that all nodes are having the same available energy and relaying nodes are sending as much packets as they can sustain with their available energy. Here the sensing nodes will only be able to utilize half of their available energy without depleting the energy storage of the relaying nodes and thus causing the network to fail. By using NC it is possible to reduce the amount of packages that each of the relaying nodes has to transmit to one packet per cycle.

For encoding the data the NC implementation of the networking framework uses a symbol size s of one bit. This means that the order of the finite field $\mathbb{F}_{2^s} = \mathbb{F}_2$ is 2 and thus the finite field only consists of two possible symbols $\{0, 1\}$. Furthermore, the encoding coefficients g_i are all equal and set to 1 to simplify the needed operations. This means that the combining of two messages M^1 and M^2 is reduced to a bitwise addition in the field $\mathbb{F}_2$, which is equal to the XOR operation. Thus it is possible to simplify the formula from equation 2.10 for combining the symbols M_k^i from different messages into the combined symbol X^k to:

$$X_k = \sum_{i=1}^{n} g_i M_k^i = \sum_{i=1}^{n} M_k^i = M_k^1 \oplus M_k^2 \oplus ... \oplus M_k^n \tag{3.5}$$

But the use of XOR operations for the encoding of packets does not only simplify the encoding part of the NC operations. One major advantage of using XOR operations for combining the packets is that the restoring of the original messages can be done the same way. For a node to be able to recover the original message M^x from an encoded message X, the node needs to have all the other messages $\{M^1, ..., M^n\} \backslash \{M^x\}$ that were combined in the message X. The original packet can then be restored by XORing each symbol X_k of the encoded message with the corresponding symbols M_k^i of the messages that were combined with the message M^x to be recovered.

Figure 2.1 already showed a simple example for the use of NC. The situation for using NC with the setup of the networking framework is a bit different since there are more than two communication paths crossing in the network. Figure 3.4 shows a 4×4 nodes network setup where all source nodes are transmitting data and NC is used in the whole network to reduce the amount of packets.

As can be seen in Figure 3.4 each of the intermediate nodes uses NC to combine the packets that it has to forward. Thus every intermediate node also has to transmit only one packet during each session, just like all the data source nodes. To be able to decode the packets again, the intermediate nodes have to combine three packets from the three nodes on the upper left side of the node before they forward the packet again. The intermediate node in the upper left corner is an exception here. Since there is no packet from the corner node this node only has to combine the packets from node A and C. To allow all destination nodes to decode their packets, it is important that all destination nodes that still have another destination node either to their right or below also broadcast their message again after they have restored it. Otherwise the other receivers are not able to restore their messages.

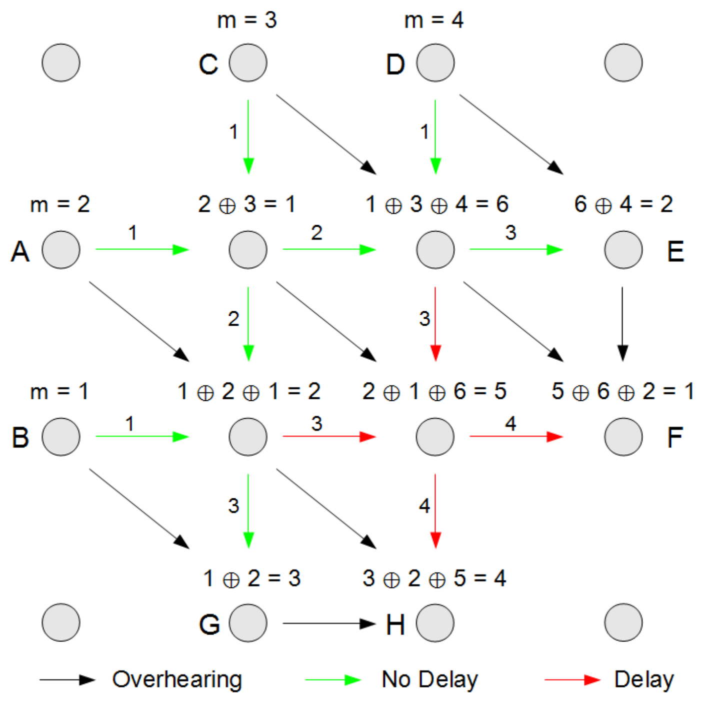


Figure 3.4: Example setup for NC with data flows and delays of the packets

But Figure 3.1 not only shows the data flows for one round of data sending with NC. For each of the four data flows from source to data sink there is also a number next to each hop of the flow. This number indicates the highest amount of hops that any of the packets combined by the sender of this hop has taken so far including the current transmission. When packets with different amounts of hops are combined, the amount of hops of the new packet will be the set to the highest amount of hops of any of the combined packets. This means that the packets on some of the data flows will be delayed since they have to wait for the arrival of packets from other data flows before they can be combined and transmitted out again. The delay of a data flow due to waiting for other packets is indicated in Figure 3.1 with red arrows for the data flow, green arrows indicate that there is no delay yet. As can be seen in Figure 3.1, this means that while the data flows for the data sources on nodes A and C will arrive at their destination without any additional delay, the packets from nodes B and D will be delayed by one additional hop. For larger networks the delay will increase, if there was another receiver to the right of node D then its data flow would already be delayed by two additional hops. This clearly indicates that the use of NC in the whole network leads to increased latency for some of its data flows to reduce the amount of necessary data transmissions.

3.3.1 Opportunistic Network Coding

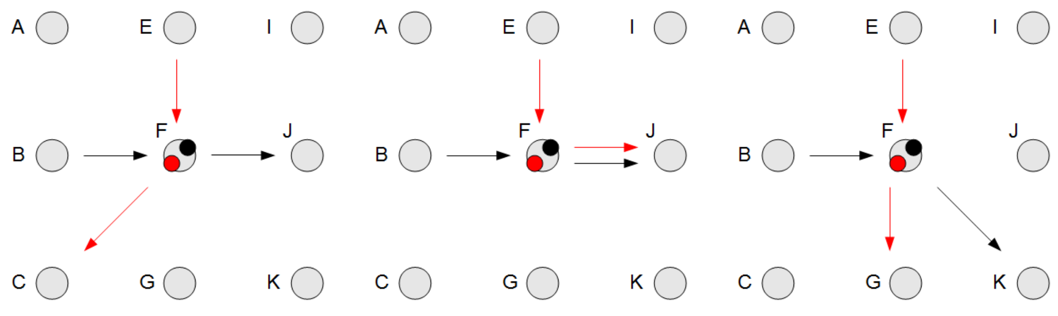
While classical NC is well suited to reduce the amount of packets needed to transmit data over the network it has one major disadvantage that can be a problem for the networking framework. Since the NC implementation does not use any energy harvesting aware mechanisms it is not possible to relieve nodes with low energy harvesting rates from parts of their transmissions to facilitate for them to maintain energy neutral operation. While this might not be a problem in networks with a quite balanced energy harvesting rate between the different nodes in the network, this can be fatal for networks with bigger differences in energy harvesting rates. Another advantage of being able to use a routing algorithm underneath ONC is the flexibility that can be provided by a routing algorithm. NC on the other hand trades in the flexibility for its reduced amount of packet transmissions.

In contrast to classical NC ONC does not replace EHAR, it is on the contrary operating on top of the routing algorithm. Due to the probabilistic component of the used EHAR algorithm a NC algorithm that is build on top of it cannot rely on a fixed traffic pattern to find possibilities for using NC in advance. Since a node needs to have to transmit at least two packets to be able to make use of NC, ONC relies on detecting these crossings in data flows between different data sources and destinations. Not every crossing of data flows is suited for using NC though, the next hops of the combined packet have to be able to recover their messages from the combined packets.

Section 2.2.3 presented several approaches for the implementation of ONC. The used ONC scheme for the networking framework is based on the COPE algorithm proposed in [31]. To be able to combine packets from different data flows at a crossing, all intended next hops of the combined data flows have to be able to overhear enough packets that each of the next hops is able to reconstruct its intended message. This means that for example a node cannot combine two packets if not at least one of the next hops of the packets is able to restore its original message from the combined packet. For the COPE algorithm this is the case when one of the receiver nodes of the combined packet is out of range of both nodes that sent one of the packets that are being combined to the current node. For the proposed ONC scheme another possible way of receiving the packets needed for restoring the original message has to be considered. The node can also receive the packet needed to restore its message from the other receiver of the combined packet. Only if this is also not possible a node is not able to combine the packets of the two different data flows.

The possibilities for ONC depend on the relative positions of the previous hop nodes that sent the packets that are being combined to the current node and the intended next hop nodes of the packets. Since the networking framework uses a fixed network setup the positions of the nodes are known. This allows to determine whether a node is able to overhear a packet or not. But the relative positions of the next hops for the combined packet can also be of importance, depending on the current data flow. There are three eligible scenarios for using ONC at a node to combine two packets.

The simplest case is when both of the next hops for the combined packet were able to overhear at least one of transmissions from the previous hop node. For reconstructing the original messages it does not matter if a node was able to overhear the message that is designated to the other next hop node or if it was able to overhear the message that it actually wants to recover from the combined packet. An example for this can be seen in Figure 3.5a. The second possible case, which is a special case of the first one, can be seen in Figure 3.5b. Here the next hop of both packets is the same node. In this case it is sufficient if the next hop node was able to overhear only one of the transmissions from the previous hop nodes.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 3.5: Possible cases for using ONC with two packets

The third and last case is a bit more complicated and does not allow an immediate reconstruction of the original packet by both receiver nodes. An example for this can be seen in Figure 3.5c. It can be sufficient if only one of the two next hop nodes is in range of one of the previous hop nodes. This is the case when the next hop node that is out of range of both previous hop nodes is able to overhear the other next hop node when transmitting its message recovered from the combined packet. However, this is only possible if the next hop node that is able to restore its message is not the destination of the restored packet because then the node would no longer send out its restored message again. One exception for combining of the EHAR packet being sent out is if it was already restored from a combined packet by the current node. Since this packet was received combined with another packet, the other nodes in the network were not able to overhear this packet. Thus all packets that were restored from a combined packet by the current node are sent out immediately without trying to combine them with other available packets. While it is also possible to combine more than two packets at a node, it is not viable for most WSNs. Not only are the scenarios that allow combining three or more packets very specific but it will also further increase the latency that is caused by ONC. If the nodes have to be able to combine three or more packets they can no longer send out a combined packet as soon as they are able to combine any two data packets.

When a node receives an encoded packet it has to determine whether it has to restore a packet from the combined packet and which of the packets that it could overhear should be used to restore the original information. For this the header of each encoded packet has two bitmaps, one bitmap indicates the origin of the packets that were combined in the message and the other bitmap determines the intended receivers for the combined packet.

Since the underlying routing scheme is probabilistic, when a node receives a packet it can never know for sure if there are more packets incoming that can be combined through ONC. If the node would just immediately forward all received packets, there would never be any chance for using ONC. Thus after receiving a packet a node has to wait some time before sending the packet out again to wait for other packets to enable NC. If the node does not have any other packets to combine with the first packet when the maximum delay timer for the first packet runs out, the packet is sent out uncombined. As can clearly be seen, a major disadvantage of the use of ONC is increased latency. Thus a tradeoff has to be found for the maximum wait time before sending out a packet uncombined that allows to combine as many packets as possible on one hand, but does not increase the latency of the packet transmission too much on the other hand.

3.4 Duty Cycling

Since the energy consumption of a sensor node where all modules are active is too high to be sustained with the power that is provided by the EHD ,duty cycling is used to periodically turn off some modules of the sensor node. Since the modules that are periodically turned off spend most of their time in idle mode, duty cycling is able to reduce the power that is wasted in idle mode while still being able to fulfill all tasks. To maximize the on time of the modules of a node while keeping the power consumption sustainable with the power provided by the EHD, the maximum sustainable duty cycle needs to be determined. This section presents the design challenges for the duty cycling implementation of the networking framework and the chosen algorithm to determine the maximum sustainable duty cycle.

One of the modules with the highest power consumption in idle mode is the radio module. While this makes duty cycling of the radio module really interesting for reducing the power consumption of a sensor node, it also causes some problems that are not present when duty cycling for example one of the sensor modules of a node. Since a node is not able to receive any packets while its radio module is turned off, the on times need to be synchronized between the nodes to allow communication. To allow all nodes in the network to communicate with each of its neighbors it is preferable to synchronize the on times of all the nodes in the network. Otherwise all the nodes in the network would need to keep track of the duty cycles of all of its neighbors and turn its radio module on according to the node that it wants to communicate with, which would lead to a computational and communication overhead for keeping track of all duty cycles. Additionally it would render the broadcasting of packets to all neighbors of a node impossible. Nodes would have to transmit a packet multiple times to allow for all its neighbors to receive the packet.

Since the duty cycle will be the same for all nodes in the network, the maximum sustainable duty cycle has to be chosen according to the energy profile of the node in the network with the least amount of available energy. Any higher duty cycle would lead to a power consumption on this node that can not be sustained by the power provided by the EHD and any power that is still available from the energy storage. To be able to determine the maximum duty cycle that is sustainable by all nodes in the network either the energy profile or the maximum sustainable duty cycle of all nodes in the network has to be known. Since distributing the duty cycle information of every node to all nodes in the network would lead to a huge communication overhead, a controller node is collecting information from all sensor nodes. This controller node can then either directly determine the maximum sustainable duty cycle and distribute it to all nodes in the network or forward the data to a base station outside the network that handles the computations and then sends the results back to the controller node again for broadcasting to the network. Since the maximum sustainable duty cycle is smaller in regard to memory than the complete energy profile of a sensor node it is preferable that each node calculates its maximum sustainable duty cycle locally and then just transmits its duty cycle to the controller node instead of its complete power profile . While this leads to a computational overhead since now every node has to calculate its own duty cycle instead of just calculating it once for the lowest energy profile on the controller node, the saved energy due to the smaller data packets outweighs the increased local computation costs. To allow the controller node or a controlling base station to react to changes in the energy profile of the network and to adapt the duty cycle accordingly all nodes have to inform the controller node if they are no longer able to sustain the current duty cycle with their changed energy profile. The controller node also has to periodically initialize an update of the duty cycle information that it has stored for all the nodes in the network to check if it is possible to increase the used duty cycle and thus also increase the possible performance of the application that is running on the nodes.

In [30] [23] Hsu, Kansal et al. proposed an adaptive duty cycling algorithm that is able to operate a sensor node in energy neutral mode. The algorithm is designed to determine and adapt the duty cycle for energy neutral operation for a singe sensor node but not for a whole network of sensor nodes. The calculation of the maximum sustainable duty cycle of each sensor node which is then transmitted to the controller node is based on this algorithm. But the adaption of the duty cycle has to be handled different for the given scenario since changes of the duty cycle depend on information from all nodes in the network.

The parameters that have to be considered for choosing the maximum sustainable duty cycle of a sensor network are the energy profile of the EHD, properties of the chosen power source for the EHD, the characteristics of the used energy storage and the average power consumption both in on and in off state. The average energy consumption of a sensor node with duty cycling is calculated by the average power consumption in on and off state weighted by the durations of the on and off times. While it is important that the average energy consumption of a sensor node during one period stays below the energy that is provided by the EHD during this period to enable longtime survivability of the node it is not sufficient to guarantee that the node never runs out of power. It is also important that a node is not able to consume energy before it is gathered. For example a node with a solar panel cannot consume the energy during the night that will be gathered by the EHD in the following day. For the energy profile of the EHD the maximum load rate and the size of the energy storage are also important, peaks in the energy profile that exceed the limits of the storage will just go to waste and have to be ignored when calculating the maximum sustainable duty cycle.

Adapting to changes in the provided energy of the duty cycling in [30] [23] is solved by changing the duty cycle during different time slots to maximize the utility of the application that is running on the nodes. While this works great for single nodes without network communication, simply changing the duty cycle for a short time slot is not preferable for larger networks of nodes since the cost of updating the duty cycle would negate the savings of a lower duty cycle or make a higher duty cycle no longer viable for energy neutral operation. Thus updating the duty cycle is only viable if the changes are big enough to overweight the communication costs that are needed for updating the duty cycle or if a node would run out of power with the current duty cycle. Since it is not viable to update the duty cycle due to small fluctuations of the energy profile it is preferable to set the actual duty cycle to a value below the maximum sustainable duty cycle of the node with the lowest energy profile. Otherwise a small decrease in the energy provided by the EHD of this node would already make an update of the duty cycle necessary or would risk that the node runs out of power and fails.

This leaves two cases when the updating of the current duty cycle is viable and may even be necessary. When the maximum sustainable duty cycle of a node in the network drops below a threshold determined by the current duty cycle, or if the maximum sustainable duty cycle of all nodes in the network is above a threshold determined by the current duty cycle. To be able to detect that a node can no longer sustain the current duty cycle, each node in the network has to regularly check if the energy profile of the EHD has changed and if the changed maximum sustainable duty cycle is above the threshold. If a node is no longer able to sustain the current duty cycle, the node has to transmit a packet to the controller node to demand a duty cycle update. The packet to the controller node has to contain the maximum sustainable duty cycle of this node which will be the foundation for a new duty cycle for the network that has to be distributed to all nodes before it becomes active. To detect if a higher duty cycle would be possible the complete duty cycle information of all nodes in the network saved at the controller node or a controlling base station has to be updated. Since this is a high energy task it is not viable to regularly adapt the duty cycle. The frequency of the update also depends on the energy gathering profile of the chosen EHD. For a solar panel for example an update every few days can be useful since the provided energy may vary quite a lot depending on the weather conditions of each day.

3.5 Low Power Listening

While duty cycling is able to reduce the energy consumption of the radio module and also the rest of the sensor node, the radio module is still constantly in listening mode during the on times of the duty cycle. This means that there is still a lot of energy simply wasted because the radio module is constantly checking the air interface for transmissions even if there are no data transmissions. Especially when only few data transmissions with long periods of time between each transmission take place, there is still a lot of energy wasted. However, just turning off the radio module when there are no data transmissions expected is also not viable because then the node is also not able to receive control messages or react to unexpected messages. With the use of low power listening a sensor node is able to reduce the energy consumed while listening for messages and still able to receive and react to messages. The networking framework will not implement its own LPL algorithm but will use the already existing implementation of B-MAC in TinyOS 2. Since B-MAC is an asynchronous LPL that does not need any time synchronization between the nodes in the network it is well suited to be used along with duty cycling.

3.6 Testing

While the previous sections focused on the design of the networking framework, this section presents the tests that will be made to evaluate the networking framework. The testing can be separated into tests showing the functionality of the networking framework on real hardware and tests performed in a simulator to evaluate the performance of the networking framework and the implemented algorithms.

The tests on hardware will be performed on a network of Mica2 sensor nodes. The amount of available sensor nodes allows to test the networking framework in a network configuration with 3×3 nodes. There are two tests that will be performed on hardware, a functionality test and power profiling of the networking framework. The purpose of the functionality test is to proof that the implemented networking framework is not only working properly in a simulator but is also running correctly on real hardware. The purpose of the power profiling is to be able to analyze the power consumption of the networking framework on real hardware.

The performance tests of the networking framework are implemented in the TOSSIM simulation environment. The performance tests can be separated into tests to evaluate the different implemented networking algorithms and the tests for the power saving strategies that are implemented. For the networking tests three modes of data forwarding will be compared, EHAR, NC and ONC running on top of EHAR. The properties that will be used to compare the different algorithms are the amount of packets that need to be transmitted for a fixed amount of data flows, the average latency for the different data flows and end-to-end packet loss for different reception strengths. Additional to the comparison of the different algorithms according to these properties, a stress test will be performed to compare the limitations of the algorithms and to analyze their performance under these conditions. Another goal of testing is to evaluate the performance of EHAR in networks with different energy profiles on the nodes in the network. The second part of the tests done in the simulation environment will focus on duty cycling. Since the available Mica2 sensor nodes are not supplied with an EHD the tests will only be done in the simulation environment. The tests for duty cycling can be separated into determining the appropriate duty cycle for a given network and adapting the duty cycle. For adapting the tests will differentiate between two cases. The first is to check whether a node appropriately informs the controller to change the current duty cycle if it can no longer sustain the current duty cycle. The second case is to update the duty cycling information that the controller has of all the nodes in the network and to check if a different duty cycle should be used. For a simpler evaluation of the performance the testing will be done with simplified values for the power that is provided by the EHD.

Chapter 4

Implementation

This chapter focuses on the implementation of the energy harvesting networking framework in TinyOS 2. The different parts of the networking framework are all implemented in separate modules that will be presented in detail in this chapter. The first section of the chapter presents an overview of the implementation which includes the structure of the framework, the interfaces that are provided by the framework and the test scenarios for the framework. The second section presents the design of the EHAR module. The third section focuses on the implementation of the NC module. The fourth section presents the implementation of the ONC module and the fifth and last section presents the implementation of duty cycling and low power listening.

4.1 Implementation Scheme

The aim of this section is to give an overview of the implementation of the networking framework which includes the structure of the implementation, the interfaces that are provided by the networking framework and also the test scenarios for the networking framework.

4.1.1 Structure

The implementation of the networking framework is segmented into several modules that are interacting with each other to be able to provide the functionality of the framework. Figure 4.1 presents an overview of the different modules of the networking framework and the existing interactions between the modules.

The central module of the implementation is the framework control. The most important part of the module is the processing of the packets that are received over the radio interface. Every received packet passes through the framework control module that distributes the packets to their appropriate modules according to the types of the packets. The framework control also provides the control interface that allows changing some of the parameters of the framework during runtime. The only other module that interacts directly with the radio interface is the send buffer module. Whenever a module wants to send out a packet over the radio interface, it passes its packet to the send buffer module that enqueues the packet in its buffer. The send buffer uses a time slotted sending scheme to avoid packet collisions between the different nodes in the network. As long as there are

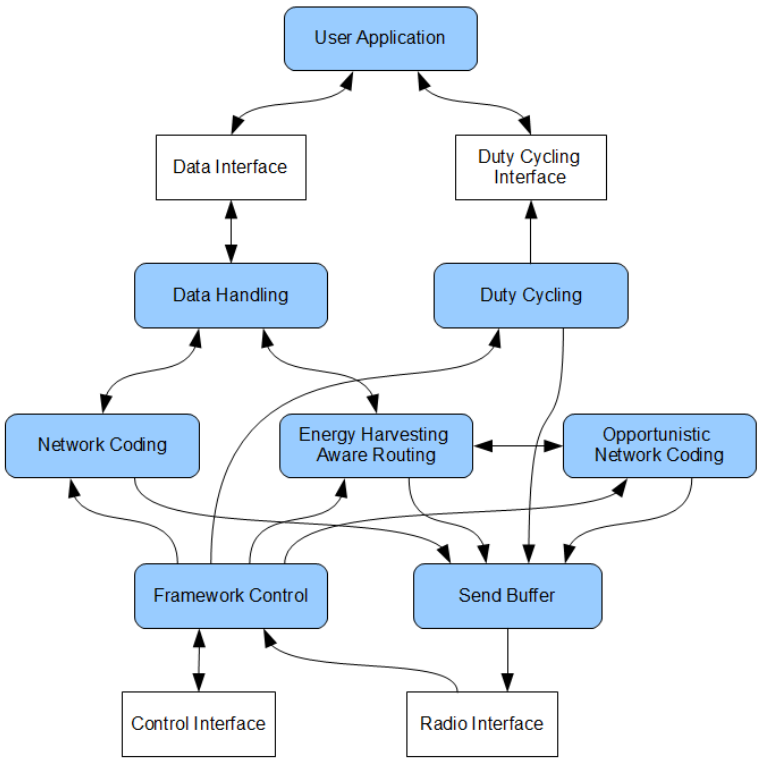


Figure 4.1: Interactions between the modules of the framework

packets in the send queue of a node the send buffer module sends out one packet during each time slot of the node.

The data module provides the data interface to the user application that is running on the nodes in the network. It is responsible for transforming data provided by the user application on the source side into actual data packets and for extracting the data from these packets again when they have reached their destination. The data module is not responsible for the actual routing of the data packets though. Depending on the chosen routing type, the data module forwards the data packets either to the EHAR module or the NC module. When the active routing module receives a packet that has reached its destination node then the module forwards the packet to the data module again. A special case is the ONC module. It is not an independent routing module but it is operating on top of the EHAR module. When ONC is activated, the EHAR module forwards its packets that it wants to send out to the ONC module instead of the send buffer module. The ONC module then tries to combine the packets before forwarding them to the send buffer module itself. When the ONC module restores a packet from a received combined packet it forwards the packet to the EHAR module again, the ONC module never interacts directly with the data handling module.

The duty cycling module is responsible for all operations that are necessary for the implementation of duty cycling on the nodes in the network. This includes not only the duty cycling of the node but also calculating the energy profile and the maximum sustainable duty cycle of the sensor node. The duty cycling module also provides the duty cycling interface to the user application that is running on the node which allows the user application to adapt its behavior to the current duty cycle.

4.1.2 Interfaces

The networking framework provides interfaces both for the user application that is running on top of the networking framework and a controlling base station that is connected to one of the nodes in the network. The functionality that is provided to the user application implemented on top of the networking framework is separated into two interfaces, the data interface and the duty cycling interface. Additionally the networking framework uses the radio interface for all communication that is taking place between the nodes in the network.

The data interface provides all methods that are needed to send and receive data on the user application. Before a user application can start sending data, the application needs to know that the networking framework has finished its setup phase and is ready to route data over the network. This is signaled to the user application with the start event of the data interface. When a user application is started and has collected some data for transmitting to a receiver node in the network it can call the sendData() method that is provided by the data interface with the data that it wants to send and the ID of the destination node of the packet. When a packet reaches its destination node then the framework forwards the data contained in the received packet and the ID of the source of the packet to the user application with the dataReceived() event.

To allow the user application to adapt its behavior to the current duty cycle of the networking framework the duty cycling interface is provided to the user applications. Every time the duty cycle used by the node is changed the dutyCycleChanged() event informs the user application of the new duty cycle. The information that is passed to the user application is the duration of a full duty cycling period and the ratio between the on period of the duty cycle and the off period of the duty cycle. Furthermore, the duty cycling interface also provides methods to the user applications to obtain the values of the duty cycle period and the duty cycle ratio that are currently on use on the node.

The controlling interface allows a base station connected to a network using the networking framework to change some parameters of the networking framework. The controlling interface allows the base station to change the data routing type used in the network. Furthermore, the controlling interface informs a connected base station of the maximum sustainable duty cycles of all the nodes in the network which allows the base station to set the duty cycle used in the network. The base station can also initiate the request to update the information on the maximum sustainable duty cycles of all the nodes in the network.

4.1.3 Test Scenarios

In order to assess the behavior of the implemented energy harvesting networking framework and to evaluate the performance of the different parts of the framework some test applications are implemented. The application SimpleDataSendingApp is just a small application that counts the amount of received data packet at each destination. The application EndToEndDelayApp is used to measure the time it takes between creating a packet on a data source and the arrival of the packet on its destination. To be able to measure the delay the message of the packet is set to the creation time of the packet which allows the receiver of the packet to calculate the time it took the packet from source to destination. The last application is called SingleDataSinkApp and is a changed version of the EndToEndDelayApp where all nodes in the network except one are data sources that send data to one data sink in the network which is the only node that is not sending data.

4.1.4 Installation

To be able to run the networking framework some prerequisites have to be fulfilled. The framework is implemented in TinyOS in version 2.1.0-2. To be able to run the networking framework without any changes the TinyOS version has to be below 2.1.1 because with this version the low power listening interface is changed. Furthermore the framework requires the Java JDK in version 1.6 and Cygwin if the networking framework shall be used on a Windows computer. Before TinyOS can be installed on a computer the native compilers and the TinyOS toolchain needs to be installed first. After the installation of TinyOS it is important that the location of the tinyos.jar file is added to the classpath and that the environment variables for TinyOS are set correctly. Detailed instructions on the installation of TinyOS can be found at [1]. To be able to run the evaluations of the networking framework Matlab needs to be installed in version R2008a or newer.

4.2 Energy Harvesting Aware Routing

This sections focuses on the implementation of the EHAR module. The functionality of this module can generally be split into two parts, the setup of the routing tables and the routing of data packets. An overview on the different states of the module and the flow of operations is given in Figure 4.2. To be able to send data forward to one of the data sinks in the network a node needs to know to which node it has to send a packet next. This means that each node in the network needs to have a routing table with entries for all the data sinks of the network. Generating these routing tables on all the nodes is the purpose of the setup phase. The second part is responsible for the actual forwarding of the data that is sent out by the data source nodes. Whenever the module receives a packet to forward it has to choose one of the entries it has for the destination of the current packet to transmit the packet towards its destination.

Each time a node wants to send a packet to one of the data sinks in the network the node first has to determine the next hop for the packet. To find the next hop for a packet, no matter if the packet originates from the node or is just being relayed, the node has to check its routing table for an entry for the destination of the packet. To be able to look for an entry in the routing table, the routing table has to be built first since at startup each node has no information about the data sinks in the network and the possible routes towards them.

Because EHAR is a probabilistic routing scheme, it is not enough to just store the routing information for the next hop that is optimal for routing a packet towards its destination. Rather each node stores all routing information that is within a threshold of

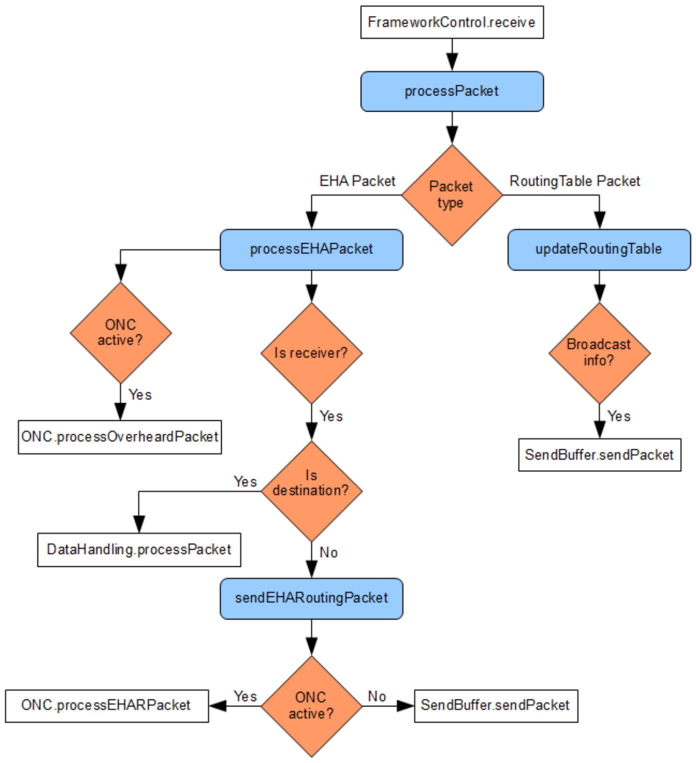


Figure 4.2: Flow diagram of the EHAR module

the routing criteria. The routing criteria here is the hop count of the shortest route from the current node to the data sink. The threshold determines the amount of hops that the minimum hop count of a next hop can be above the lowest minimum hop count of all next hops to a destination from the current node. The routing information that is stored for each of the entries in the routing table consists of five elements. The first element is the ID of the data sink that the routing information is for. The second element is the ID of the next hop that a packet to the data sink is sent to. The third element is the hop count that indicates the minimum amount of hops that a packet needs to the data sink when it is sent via the next hop of the routing information. The fourth element is the energy level of the next hop node. The energy level of the node is necessary to choose the next hop when there are multiple entries for a data sink. The higher the energy level of a next hop, the higher is the probability that the node is chosen as next hop. The fifth and last element is the sequence number of the routing information. Sequence numbers are added to the routing information to be able do differentiate between old and new routing information when updating the routing information tables.

The construction of the routing tables is initiated by the data sinks in the network. Each node that wants that all the other nodes in the network are able to route data towards them has to start the setup phase for the routing tables by broadcasting its own routing information. For this start packet the base ID and the next hop ID are the same, they both contain the ID of the node that is sending out the packet. The energy level is set to the energy level of the node and the hop count is set to one since all nodes that receive this packet can directly transmit data to this node. Each node has a sequence number for its routing information. If the data sink sends out its routing information for the first time the sequence number of the routing packet can be set to an arbitrary value, if the data sink wants to update its routing information then the sequence number needs to be set to a value greater than the sequence number of the last routing information that was broadcasted by this data sink.

Each time a node receives a routing information packet, it has to determine whether the received routing information needs to be stored in the routing table or not. The first check the node has to make is to determine if it already has an entry for the data sink that sent out the received routing information packet or not. If there is no entry yet for the ID of the data sink, the routing information is entered into the routing table. Since it is the first routing information for this data sink, the hop count of the received routing information will now also be the current minimum hop count for transmitting data to this data sink.

If the node already has an entry for the ID of the data sink then the next step is to check if the node already has an entry for the next hop of the routing information. If the node does not yet have an entry for this next hop for the data sink of the routing information then the node has to compare the hop count of the routing information to the minimum hop count of all the entries in the routing table for this data sink. If the hop count of the routing information is within the threshold of the minimum hop count then the routing information is added to the routing table, otherwise the information is discarded. If the node already has an entry for the next hop then the hop count is compared to the hop count of the entry that is already in the routing table. If the new hop count is lower than the hop count in the routing table the already stored entry in the routing table is replaced with the new one. A special case is when the sequence number of the received routing information is higher than the already stored routing information for the data sink. This indicates that the data sink initiated a routing table update. In this case the node first deletes all its entries for the data sink of the received routing information and then adds the received routing information to the routing table as if it were the first entry for this data sink.

To be able to distribute routing information in the whole network, each node also has to broadcast routing information again so the other nodes in the network can construct their routing tables. Not every routing information that is added to the routing table also needs to be broadcasted again afterwards. Routing information is only broadcasted again if it is the first entry in the routing table for a data sink or if there was already an entry in the routing table for the data sink and the hop count of the new routing information is below the old minimum hop count for this data sink. Some of the values of the received routing information packet need to be updated first though before the routing information packet is broadcasted again. While the ID of the data sink and the sequence number stay the same is the ID of the next hop changed to the ID of the current node. Also the energy level has to be set to the energy level of the current node and the old hop count is increased by one. This new routing information is then broadcasted out for the other nodes to create their routing tables. When none of the nodes in the network are broadcasting routing information anymore then the routing tables on all nodes should be complete.

The second part of the EHAR module deals with the actual routing of the data packets towards their destinations. Since the protocol uses no fixed routes each time the module is sending out a packet is has to choose a next hop first. To achieve a balanced spreading of the traffic according to the available power of the next hop nodes the probabilities for choosing an entry in the routing table as next hop depend on the energy level of the next hop nodes in the routing table entry. Thus a node with a higher amount of available power will route more packets over time than a node with a low amount of available power.

When the EHAR module receives a packet the first thing it does is updating the energy level in the routing table entries for the sender with the new energy level from the packet. The next step a node does is to check if the packet is actually intended for the current node or not. If the packet is really for the node and was not just overheard then the next check is whether the current node is the destination of the received packet. If this is the case then the packet is forwarded to the data handling module which is responsible for the data interface that is provided to the user applications. If the current node is not the destination of the packet then the module has to check if it has an entry in its routing table for the destination of the packet. If there is an entry in the routing table for the destination then the node has to select the next hop for the packet from the possible next hops that it has stored for this destination. Listing 4.1 shows how a node picks the next hop for a data packet.

```
for(i = 0; i < num_receivers; i++){
  if(routing_table[i][0].base_id == receiver_id){
    for(j = 0; j < NUMBER_ENTRIES; j++){
      if(routing_table[i][j].next_hop_id != 0){
        total_energy = total_energy + routing_table[i][j].energy_level;
      }
    }
    rand_nr = call Random.rand16();
    rand_nr = rand_nr % total_energy;
    total_energy = 0;
    for(j = 0; j < NUMBER_ENTRIES; j++){
      if(routing_table[i][j].next_hop_id != 0){
        total_energy = total_energy + routing_table[i][j].energy_level;
      }
      if(rand_nr < total_energy){
        address = routing_table[i][j].next_hop_id;
        if(address == sender_id && sender_id != 0){
          if(j > 0){
            address = routing_table[i][j-1].next_hop_id;
          }else{
            address = routing_table[i][j+1].next_hop_id;
          }
        }
        break;
      }
    }
  }
}
```

Listing 4.1: Get the next hop for a packet from the routing table

Before being able to choose, the node has to find the entries for the ID of the destination of the data packet. If an entry is found then the node has to add the energy levels of all the entries it has for the ID of the destination. This is needed to be able to choose one of the entries. Since the module wants to pick one of the entries according to their probabilities and not always pick the same entry or just cycle between them, a random element is needed. Thus in the next step a random number is generated that is the foundation for choosing the next hop. This random number is taken modulo the sum of the energy values of all the entries for the destination to reduce the random number to a value between 0 and $total\_energy - 1$. Now to pick the entry in the routing table according to the random number, the sum of all energy values is calculated again. But this time the addition stops when the sum of the energy values exceeds the value of the random number. The entry where the sum exceeds the random number is the entry in the routing table that is chosen for the next hop.

A special case is when the ID of the sender, that sent the packet to the current node, matches the ID of the chosen next hop. Since this would mean that the packet is transmitted in a circle, the next hop needs to be changed. If the chosen next hop is not the first entry in the routing table, then the entry before the chosen entry is selected as next hop, otherwise the entry after the chosen entry is selected as the new next hop. And there is another special case for the selection of the next hop. Since the hop count of a routing information only needs to be within the threshold of the minimum amount of hops towards the data sink, each routing packet also has an add_hops field that indicates the maximum amount of additional hops that a packet is allowed to take on its way to the data sink. If the chosen next hop is above the minimum hop count then the value is decreased by one. When the next hop for a packet is selected and the add_hops field is already 0 then next hop can only be set to one of the possible next hops that can reach the data sink within the minimum amount of possible hops for this node. When the module was able to successfully find a next hop for the packet, the routing values of the data packet are changed, the energy level field of the packet is set to the energy level of the current node and the packet is forwarded to the sending buffer. If the node was not able to find an entry for the destination of the packet in its routing table then the packet is discarded since the node is not able to successfully transmit it closer to its destination.

4.3 Network Coding

In this section the implementation of the NC module is presented. Unlike the EHAR module the NC module does not build up a routing table before being started since the positions of the nodes in the network are fixed and there is no additional information about the nodes in the network needed by the NC implementation. The functionality of the module consists of the processing of received packets which includes combining data packets, restoring data from combined packets and requesting and processing retransmits of data packets. An overview on the different states of the module and the flow of operations is given in Figure 4.3.

When the NC module receives a NC packet by the framework control then the first step is to check the position of the sender of the packet in relation to the current node. If the received packet is not from one of the three nodes that are located to the left, the top

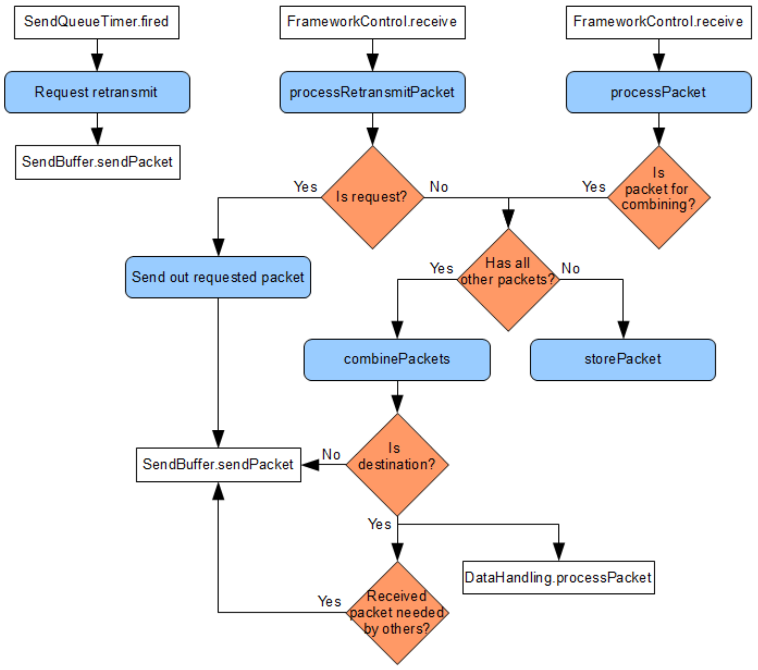


Figure 4.3: Flow diagram of the NC module

or the top left of the current node then the packet is discarded. The next step for a node is to check how many of the packets to be combined already have been received. Because the node is only able to construct its new packet and send it out again if it has received the packets from all the data flows that shall be combined. Normally a node needs to have the packets from the nodes at the three mentioned positions to continue, only the most left lower receiver node, the most upper right receiver node and the node most upper left inner network node need the packets from just two nodes to continue. If the node does not yet have all the required packets to continue then the received packet is stored in a buffer.

When a node has received all the required packets it can combine the received packets. But not all fields of the received packets are combined, only the source ID, the destination ID and the message are actually combined by the NC module. If the current node is one of the data sink nodes in the network then the new combined source ID and the new combined message actually contain the data intended for this data sink, this data is then forwarded to the data handling module. Afterwards the new combined packet is forwarded to the send buffer module which broadcasts the packet again except if the current node is the most right lower receiver or the bottom right receiver node.

Each time a packet is added to the buffer of the received packets the NC module also starts a timer. This timer is used to monitor the amount of time that the packet has already spent in the buffer. The value of the timer is chosen according to the expected amount of time until all packets that should be combined should have arrived at the current node. When this timer fires the node sends out a retransmit request to all the nodes whose packets are still missing, since this means that these packets have most likely been lost due to noise. When a node receives a retransmit request it checks its sent buffer if the requested packet has already been transmitted. If this is the case then the packet is retransmitted.

4.4 Opportunistic Network Coding

The following section presents the implementation of the ONC module. The implementation of ONC can be separated into three main parts. The first part is the processing of packets that the underlying EHAR algorithm wants to transmit. The goal is to find possibilities to combine some of the packets and save transmissions. The second part deals with all overheard packets. These packets need to be stored to be able to restore combined packets again on the receiver side of their transmission. The third part is responsible for processing all received packets that were encoded with ONC. The goal of this part is to correctly restore the original packet that is intended for the current node from the combined packet and to request packet retransmits if necessary. Figure 4.4 gives an overview of the implementation of the ONC module and the flow of event ins the module.

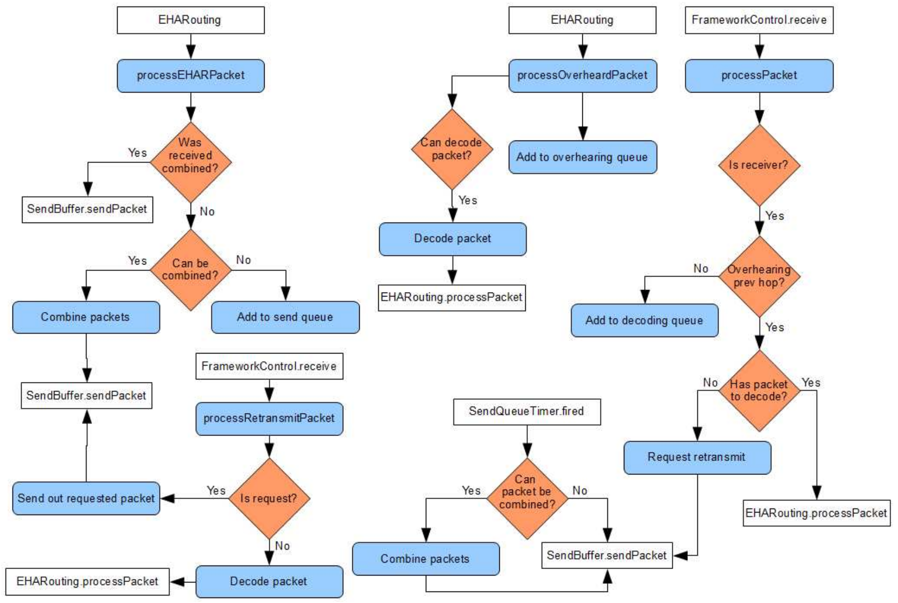


Figure 4.4: Flow diagram of the ONC module

The first part of the ONC module is dealing with the packets received from the EHAR module. When ONC is enabled, each packet that is sent out by the EHAR module passes through the ONC module before it is sent out. To be able to detect possibilities for ONC, the packets have to be stored in a buffer before they are sent out. But each packet is only stored in the buffer for a limited time, either until the packet can be combined with another packet or the packet exceeds its maximum time in the buffer. If a packet in the buffer exceeds its maximum delay time the packet is sent out even if there is no possibility to combine the packet. When the maximum delay time of a packet is reached or a new packet is added to the buffer the ONC module checks if there is any possibility to combine the packets in the buffer. When a possibility exists the combined packet is sent out immediately and the two packets that were combined are removed from the buffer. The third reason for sending out a packet is when the buffer is full. Then the oldest packet in the queue is sent out and removed from the queue, even if the packet has to be sent out uncombined. Each packet that is sent out uncombined by the ONC module is added to its sent out queue to be able to react to retransmit requests from other nodes. If there is already an entry in the sent out queue for the same next hop, the old entry is deleted first since the node can not distinguish between those two messages in case of a retransmit request.

To determine if two packets can be combined the current node needs to check the position of the senders that transmitted the packets to the current node and the intended receivers of the packets when transmitted again by the current node. Listing 4.3 shows the simplest case that allows the combining of two packets. If the receiver of the second packet is within receiving range of the sender of the first packet, the receiver of the second packet is able to restore its intended packet from the combined packet. If the receiver of the first packet is also within receiving range of the sender of the second packet, both receivers are able to restore their packets from the combined packet. This means that the current node can combine the two packets that it needs to send out.

```
if(checkForOverhearing(sender_1, receiver_2) == TRUE){
  if(checkForOverhearing(sender_2, receiver_1) == TRUE){
    //Both receivers can overhear the other sender
    combine = TRUE;
  }
}
```

Listing 4.2: Simplest case for combining two packets from the buffer

However, the example from listing 4.2 only covers the simplest possibility that allows the combining of two packets. To further increase the possible gain from using ONC on top of EHAR there are more cases that have to be checked since they also allow combining two packets. In listing 4.3 all possible combinations are checked that allow the combining of packets if the second receiver is able to overhear the packet from the first sender.

The first case that is checked is if the first receiver is also able to overhear the second sender just like in the simple example from listing 4.2. If the first receiver is not able to overhear the second sender the next check that is made is whether the first receiver is able to overhear the packet from the first sender. If this is the case then it is also possible to combine the packets since the first receiver can just use the information from the overheard packet from sender one instead of having to restore it from the combined packet. If both other cases are not fulfilled, there is one more possibility that allows combining the packets if the second receiver is within range of the first sender. Since the first receiver is out of range from both senders it has to receive the packet needed for restoring its packet from the other receiver node. But here it is not enough that the receiver nodes are within range of each other. It is also necessary that receiver two is not the destination of its packet because then receiver two does not send out again the packet needed by receiver one to restore its packet.

```
if(checkForOverhearing(sender_1, receiver_2) == TRUE){
  if(checkForOverhearing(sender_2, receiver_1) == TRUE){
    //Both receivers can overhear the other sender
    combine = TRUE;
  }else if(checkForOverhearing(sender_1, receiver_1) == TRUE){
    //Both receivers can overhear at least one sender
    combine = TRUE;
  }else if(checkForOverhearing(receiver_2, receiver_1) == TRUE){
    //One receiver can overhear a sender and
    //both receivers can overhear each other
    if(receiver_2 != packet_2.destination){
      //Only combine the packets if the next hop that needs to
      //retransmit the packet is not the destination of the packet
      combine = TRUE;
    }
  }
}
```

Listing 4.3: Try to combine two packets from the buffer

While these three cases cover all possibilities that allow to combine two packets if the packet from the first sender can be overheard by the second receiver, these are not all possible combinations yet. The same checks need to be made when the packet from the second sender can be overheard from the first receiver. If none of the receivers is able to overhear the sender of the other packet, it might still be possible to combine the two packets if the receivers are able to overhear the senders of their packet. If both receivers are able to overhear the transmission from the sender of their own packet then the two packets can be combined. If only one receiver is able to overhear the packet from its own sender then it is still possible to combine the packets, provided the receivers can overhear each other and the receiver that is able to overhear its sender is not the final receiver of its packet. If those conditions are fulfilled the receiver will transmit the packet that it received again and the other receiver is able to overhear the transmission and use the packet to restore its packet from the combined packet.

But before being able to send out a combined packet, the ONC packet must be generated. For the fields message, source, destination and sequence number of the packets combining is quite simple. The data of these fields can be combined with a simple XOR operation of the entries of the two packets that are being combined. For the combined_packet.add_data field it is not enough to simply combine the entries of the two packets. The combined_packet.add_data field is used to determine which of the combined packets is destined to which receiver of the combined packet. This is achieved by combining the ID of the receiver with the smaller ID with the ID of the source of the packet for this receiver. The previous hops that sent the packets to the current node determine which packets have to be used to restore the original packets. Each bit of the combined_packet.previous_hops represents one of the eight possible neighbors of the current node. If the previous hops are the same for both packets only the bit for this previous hop is set. Otherwise the according bit is set to 1 for both of the previous hops . The same is done with the next hops of the two packets that are needed to determine if a node has to try to restore a packet from the combined packet or not. The energy level field of the new packet is set to the energy level of the current node and the add hops field is set to the lower value of the field from the two packets being combined. This has to be done because the node cannot differentiate which of the add hops fields is for which next hop since the value of the field could have changed on the current node. At last, the type of the new combined packet is set to recognize the packet as an ONC packet.

Not every data packet sent out by the EHAR algorithm is eligible to be combined by the ONC module. There are two exceptions that prevent the combining of a packet. The first reason to not encode a packet is if the current node already had to restore the current packet from a combined packet. This means that the other nodes that would have been able to overhear the packet if it was transmitted uncoded have not been able to receive it and thus can not use the packet for restoring the original messages again. The second reason is a special case of the first reason. If the other receiver of the combined packet that the current packet was restored from needs this packet to be able to restore its part of the combined packet then the packet can not be sent out encoded to guarantee that the other receiver is able to successfully receive it and restore its packet. In both cases the packets are sent out immediately without adding them to the sending buffer of the ONC module to check for coding possibilities.

To be able to successfully restore messages from combined packets each node needs to have at least one of the EHAR packets that were combined in a packet. The second part of the ONC module deals with the storage of overheard EHAR packets that are needed for restoring combined packets. However, to be able to restore packets for possibilities that allow the combining of packets, it is not enough to just store all overheard packets in the buffer. Each node also needs to store all EHAR packets that were either sent out or received by the node in the overhearing buffer. The packets that are sent to the current node also include the packets that the node was able to restore from combined packets.

To determine if there is a packet in the overhearing buffer that can be used to restore a packet from a combined packet two parameters need to be checked to see if it is the correct packet. These parameters are the sender of the packet in the overhearing queue, and the receiver of the packet. But this means that if there is more than one packet in the queue that has the same sender and receiver, there is no way to determine which of the two packets is the correct packet for restoring information from a combined packet. Thus before adding a new packet to the overhearing buffer, for each entry in the overhearing buffer it is checked if the entry has the same sender and receiver as the packet that is to be added to the overhearing queue. The code for this can be seen in listing 4.4. If this is the case for an entry all data of this entry is set to 0.

When a node was not able to restore its packet from a received combined packet, the packet is added to the decoding buffer of the ONC module. Before adding an overheard packet to the overhearing buffer a node has to check if there are currently entries in its decoding queue and if the packet that it just received is the packet needed for restoring the packet of an entry.

The third part of the ONC module deals with received combined packets. Not every ONC packet received by a node is actually intended to be restored at that node. Before a node tries to restore its intended packet from a received combined packet it has to determine if it actually has to do anything with the received packet. If the current node is one of the intended receivers it then has to check whether it can restore the original packet with one of its overheard packets, it has to wait for the transmission of a packet from the

```
for(i = overhearing_head; i < overhearing_head + overhearing_size; i++){
  entry = (i % ONC_QUEUE_SIZE);
  if(overhearing_queue_senders[entry] == sender_id){
    if (overhearing_queue[entry].receiver_id == msg->receiver_id){
      //Entry in the queue with same sender_id and receiver_id
      //Can't differentiate between those packets thus set old entry to 0
      overhearing_queue_senders[entry] = 0;
      overhearing_queue[entry].receiver_id = 0;
      overhearing_queue[entry].source_id = 0;
%     }
%   }
%}
```

Listing 4.4: Check for entries in the overhearing buffer matching the current packet

other receiver to restore its packet or it has to request a retransmission of a packet because it has no entry to be able to decode the packet.

When the ONC module receives a combined packet, the first step is to check whether the current node is one of the intended next hops of the packet. If this is not the case the packet is discarded, or else the node determines the ID of the other next hop of the packet. The ID of the other next hop is needed for two reasons. First to determine if the other node needs the packet from the current node to restore its packet from the combined packet. Second to determine which of the two packets that were combined in the received packet is intended for the current node.

To be able to determine which of the two packets is the intended packet a node first has to find a packet in its queue of overheard packets that could be used to restore a packet from the combined packet. Before searching the queue of overheard packets the node determines the data to check if the packet can be used for decoding. Thus the node first checks the previous hops of the received packets. If a previous hop is in overhearing range of the current node the node checks whether it has an entry in its overhearing queue that can be used to decode the packet. For a packet to be eligible for decoding the sender and receiver of the overheard packet have to match the previous hop and the sender of the combined packet. This does not guarantee though that the packet is actually the correct packet to restore information from the combined packet.

To be able to continue decoding the packet the node has to determine which of the two packets that are combined in the received packet it is intended to receive and further process. This also allows to check if the current packet from the overhearing queue is actually suitable to restore information from the combined packet. Listing 4.5 shows the code to determine the correct packet. First the node has to determine if its ID is smaller than the ID of the other next hop node or not. The data that is encoded in the add_data field of an ONC packet consists of the ID of the next hop with the smaller ID and the ID of the source of the packet for this ID. This field is used to determine whether the node can recover the data from the combined packet by using XOR on the data from the overheard packet and the combined packet or if the node can use the data directly from the overheard packet since the overheard packet already contains the data of the packet that is intended for the current node.

```
if(other_next_hop_id < TOS_NODE_ID){
  smaller_id = other_next_hop_id;
  id_is_smaller = FALSE;
}
check_value = msg->add_data ^ smaller_id;
if(check_value == (msg->source_id ^ overheard_packet.source_id)){
  if(id_is_smaller == FALSE){
    use_overheard_packet = TRUE;
  }
}else if(check_value == overheard_packet.source_id){
  if(id_is_smaller == TRUE){
    use_overheard_packet = TRUE;
  }
}else{
 //The message is not correct for restoring
  correct_packet = FALSE;
}
```

Listing 4.5: Determine how to restore the original message with the overheard packet

To determine this, a check value has to be generated first by using XOR on the add_data field of the received combined packet and the smaller ID of the two next hops of the packet. This means that the check value now matches the source of the packet that is intended for the node with the smaller ID. If the XOR combined source IDs of the combined packet and the overheard packet match the check value then the node has to use XOR if it has the smaller ID or just the overheard packet if the other next hop has the smaller ID. If the check value matches the source ID of the overheard packet then the node has to use the data from the overheard packet if it has the smaller ID or combine the overheard packet with the received combined packet if the ID of the other next hop is smaller. If the check value does not match any of those two conditions then it is not suitable to recover data from the received ONC packet.

If the overheard packet is suitable for restoring and the node has determined which of the two packets that were combined it is intended to receive it can actually restore its data. Listing 4.6 shows the code to restore the data from the combined packet by using XOR on the received packet and the overheard packet. While restoring of source, destination, message and sequence number is achieved by using XOR the entries of the other data fields of the restored packet are not restored from the combined packet. The value of the is_combined field is used as a helper to forward information to the ONC module for when the packet is sent out again. If the packet is needed by the other next hop to restore its data from the combined packet then the field is set to the ID of the other next hop. If the packet is not needed for restoring the field is set to 255. This indicates to the ONC module that this packet was restored from a combined packet and is not eligible for ONC when it is sent out again by the node. Furthermore, the type of the packet is set to EHAR, the energy level and the add hops fields are set to the values from the combined packet and the value of the add_data field is set to the ID of the current node since the EHAR module uses this field to store the receiver of a packet. This restored packet is then forwarded to the EHAR module for further processing.

```
restored_pkt.source_id = msg->source_id ^ overheard_packet.source_id;
restored_pkt.destination = msg->destination ^ overheard_packet.destination;
restored_pkt.message = msg->message ^ overheard_packet.message;
restored_pkt.seq_nr = msg->seq_nr ^ overheard_packet.seq_nr;
if(needed_for_other_next_hop == FALSE){
  restored_pkt.is_combined = 255;
}else{
  //Set is_combined to the ID of the hop that needs this packet for decoding
  restored_pkt.is_combined = hop_id;
}
restored_pkt.type = MSG_TYPE_EHA_ROUTING;
restored_pkt.add_data = TOS_NODE_ID;
```

Listing 4.6: Restore data from a combined packet

One special case is when the current node is actually the receiver of both packets that were combined. Then the node also has to restore the second packet and forward it to the EHAR module. If none of the previous hops of the received encoded packet are in range of the current node then the node is not able to immediately restore the data since it needs the restored packet from the other next hop to restore its data. If this is the case then the packet is added to the decoding queue. If one of the previous hops is in range of the current node and the node still has no packet in its overhearing queue to restore its message from the combined packet, the packet is added to the overhearing queue and the node requests a retransmit of the packet that it needs to restore its packet from one of the previous hops in range of the node. The node that receives the retransmission request checks if it has an entry in its sent out queue where the receiver is the sender of the combined packet and transmits the packet again if it is found. When the node receives the retransmission of the requested packet, it tries to recover its message from the combined packet with the received packet. If the packet is not correct to restore its message this means that the previous hop has already overwritten the packet needed for the restoration in its sent out queue.

4.5 Duty Cycling

This section presents the implementation of the duty cycling module of the networking framework that periodically turns off the radio module to reduce the power consumption on all sensor nodes in the network. An overview of the flow between the different methods of the duty cycling module is given in Figure 4.5. Furthermore, this section presents the details for LPL which uses the existing implementation in TinyOS 2.

The functionality of the duty cycling module can be separated into two main parts, determining and distributing the maximum sustainable duty cycle that can be sustained by all nodes in the network and the actual duty cycling of a node. The tasks of the first part not only consist of determining and distributing an initial duty cycle, but also include checking locally if a node can still maintain the used duty cycle or if a higher duty cycle would be possible. Since the used sensor nodes do not actually possess an energy harvesting device the duty cycling module also implements a simple simulation of an energy harvesting device and the power consumption of the sensor node.

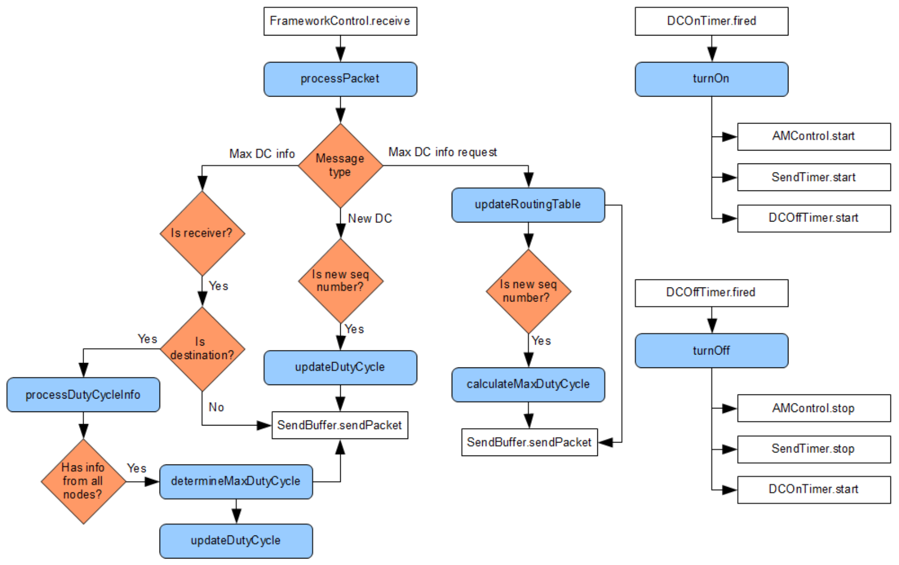


Figure 4.5: Flow diagram of the duty cycling module

The most important part of the module is calculating the maximum sustainable duty cycle that is made on all nodes in the network as can be seen in listing 4.7. The maximum sustainable duty cycle of the individual nodes is needed to determine the duty cycle for the whole network that allows the networking framework to run the network without exceeding the available power in parts of the network. The maximum sustainable duty cycle of a node depends both on the amount of energy that is provided by the energy harvesting device, and the power that is consumed by the node in on and off mode of the duty cycle. The power provided by the EHD and the power consumption of the nodes are simulated by the implementation. The power consumed in off mode of duty cycling is set to 0 to reduce the complexity. Furthermore, it is assumed that the radio module is the biggest power consumer of a node and that simple sensor readings and computations can be ignored. More complex sensor readings like a video camera would have to be added to the calculations.

The power consumed by a node is influenced by four main factors: the cost of sending a packet, the cost of receiving a packet, the cost of overhearing a packet and the cost of an active radio module. The cost of the active radio module is not a fixed value but depends on the time that the radio module is actually turned on. The values for these four factors that are used in the implementation are taken from hardware measurements of a Mica2 sensor node. When duty cycling is activated the node periodically calculates the power that was consumed during the last period and the mean value over several periods then indicates the average power consumption of a node with the current duty cycle. From the average power consumption during an update period the average power consumption for one duty cycling period is calculated. But at the start, there is not yet any duty cycle running and there are no values for the average power consumption of the node. Thus the duty cycling module calculates the worst case energy consumption of a node during a duty cycling active period. For the worst case it is assumed that the node sends out a packet at each of its sending time slots during the active period and that the nodes either overhear or receive a packet from all of its neighbors at all of their sending time slots. The costs of overhearing and receiving of a packet are averaged in case the values are different. Since the energy consumption also depends on the packets received from the neighboring nodes the power consumption of nodes on the outer edges of the network will be lower than of those in the middle of the network.

```
max_energy_used = SENDS_PER_DC*SENDING_COST + SENDS_PER_DC*num_neighbors*
    (RECEIVING_COST+OVERHEARING_COST)/2 + dc_on_time/RADIO_MODULE_ON_COST;
for(i = 0; i < ENERGY_TABLE_SIZE; i ++){
  avg_energy_gathered = avg_energy_gathered + energy_gathered_table[i];
  avg_energy_used = avg_energy_used + energy_used_table[i];
  if(energy_used_table[i] != 0){
    energy_used_entries++;
  }
}
if(energy_used_entries > 0){
  //Calculate average energy used for one duty cycle
  avg_energy_used = avg_energy_used/energy_used_entries;
  avg_energy_used = avg_energy_used*dc_on_time*(off_cycles + 1);
  avg_energy_used = avg_energy_used/ENERGY_UPDATE_INTERVAL;
}else{
  //There is no data from duty cycling yet
  //Calculate duty cycle for worst case estimate
  avg_energy_used = max_energy_used;
}
avg_energy_gathered = avg_energy_gathered / ENERGY_TABLE_SIZE;
//Calculate the ratio between the gathered energy and
//the average energy used in one duty cycle period
energy_ratio = (float) avg_energy_gathered / (float) avg_energy_used;
duty_cycle_period = (float) ENERGY_UPDATE_INTERVAL / energy_ratio;
//Calculate the duration of the sleep time in multiples of the on time
new_sleep_cycles = duty_cycle_period / (float) dc_on_time - (float) 1;
```

Listing 4.7: Calculation of the maximum sustainable duty cycle

To reduce the influence of short fluctuations of the power provided by the EHD but still being able to adapt to long time changes in the provided power the gathered power is averaged over several time slots. To be able to already calculate a duty cycle at the start of the node the buffer that is used to store the gathered energy for each time slot is initialized with the expected values of the gathered energy for the simulation. For the calculation of the duty cycle the ratio between the harvested power during an update period and the average power consumption for one duty cycling period is calculated first. The result is then used to determine the duration of one duty cycling period for the maximum sustainable duty cycle. Since the on period of the duty cycle is fixed the factor for the ratio between the sleep period and the active period of the duty cycle can be calculated as well.

Before the duty cycling module can start with duty cycling the radio module the maximum sustainable duty cycle of the network needs to be determined, which is started by the controller node. This node also provides an interface to a connected base station. The controller nodes broadcasts a request over the network for all the nodes in the network to send their maximum sustainable duty cycle to the controller node. When the controller node has received the maximum sustainable duty cycle of all nodes in the network it can either determine the duty cycle for the network locally or forward the data to the base station to perform the calculations. If the controller node determines the duty cycle locally then the new duty cycle for the network is the worst duty cycle that was received from any of the nodes in the network. This broadcast is then started again to the network together with a start time for the duty cycle. The start time is used to ensure that all nodes in the network will start using the new duty cycle at the same time.

Since the power provided by the EHD or the power consumption of a node can change over time it might be necessary to update the used duty cycle. There can be two reasons for updating the used duty cycle, either because a node can no longer sustain the current duty cycle or because all nodes in the network could sustain a higher duty cycle. To determine if the nodes can still sustain the current duty cycle all nodes periodically calculate their maximum sustainable duty cycle. If a node can no longer sustain the current duty cycle and the energy that is stored in its energy buffer is below the update limit then it sends its new maximum sustainable duty cycle to the controller node. If there is a base station controlling the duty cycling the new duty cycle is forwarded to the base station. Otherwise the controller node broadcasts the new maximum sustainable duty cycle in the network with a corresponding start time that guarantees that all nodes have received the new duty cycle until then. To determine if all nodes in the network could sustain a higher duty cycle the controller node periodically initiates a request for all nodes to send their current maximum sustainable duty cycle to the controller node like it is done for setting the initial duty cycle. This data is then either forwarded to a base station or the controller node itself checks if the current duty cycle needs to be changed and distributes the new maximum sustainable duty as well as the start time for the new duty cycle to all nodes in the network.

```
//Turn on the radio module
call AMControl.start();
if(call SendTimer.isRunning() == TRUE){
  //Stop the current SendTimer to be able to start it synced with the DC
  call SendTimer.stop();
}
//Start the send timer of the send buffer module
start_time = call SendTimer.getNow() + TOS_NODE_ID*WAIT_MULT + TIMER_OFFSET;
call SendTimer.startPeriodicAt(start_time, NETWORK_SIZE*WAIT_MULT);
if(call DCTimerTurnOff.isRunning() == FALSE){
  call DCTimerTurnOff.startOneShot(dc_on_time);
}
if(call EnergyUsageTimer.isRunning() == FALSE){
  call EnergyUsageTimer.startPeriodic(ENERGY_UPDATE_INTERVAL);
}
```

Listing 4.8: Starting the active period of the duty cycle

The second part of the module is responsible for the actual duty cycling of the nodes of in the network. The duty cycling on a node is started when the node receives its first duty cycle. If a node receives a new duty cycling update when duty cycling is already activated the duty cycle needs to be changed accordingly. Since the duty cycling turns off the nodes' radio module it is necessary that the active and sleep phases of the duty cycles of all nodes in the network are synchronized to allow radio communication during the nodes active phases of the duty cycle.

The basic behavior of the duty cycling can be separated into firing the on timer and firing the sleep timer. When the on timer of the duty cycle is fired, the node has to activate all necessary modules of the sensor node and the networking framework to enable the correct operation of the node. The behavior when the on timer is fired can be seen in listing 4.8. At first the duty cycling module has to turn on the radio module to enable receiving and sending of packets on the node again. Afterwards the module also has to start the send timer of the send buffer module again for starting the actual transmission of the packets that the node queued in its send buffer while the duty cycle was in sleep mode. If all modules are started again the timer is started for starting the sleep period of the duty cycle. At the end the node checks if the timer for periodically updating the power consumption is already running and starts the periodic timer if not.

```
//Turn off the radio and stop the send timer
if(call SendTimer.isRunning() == TRUE){
  call SendTimer.stop();
}
call AMControl.stop();
//Check if there is a new duty cycle
timer_offset = call SendBuffer.getTimerOffset();
if(call LocalTime.get() >= update_time-timer_offset && update_time != 0){
  duty_cycle = updated_dc;
  update_time = 0;
  if(call EnergyUsageTimer.isRunning()){
    call EnergyUsageTimer.stop();
  }
  //Delete all entries in the energy_used_table and short_energy_used
  //since they are for the old DC
  short_energy_used = 0;
  for(i = 0; i < ENERGY_TABLE_SIZE; i++){
    energy_used_table[i] = 0;
  }
  energy_used_current_entry = 0;
}
if(call DCTimerTurnOn.isRunning() == FALSE){
  call DCTimerTurnOn.startOneShot(dc_on_time*duty_cycle);
}
```

Listing 4.9: Starting the off period of the duty cycle

When the sleep timer for the duty cycle fires its time to turn the modules off again that were started when the on timer for the duty cycle fired as it can be seen in listing 4.9. First the duty cycling module deactivates the send timer of the send buffer module to stop all transmissions by the node. Then the module can turn off the radio module which will make it impossible to receive or transmit packets on the node during the duty cycling sleep period. Before the module starts the timer for the active period of the duty cycle the module has to check if it needs to update the current duty cycle. If there is a start time for a new duty cycle and the current time is larger than the start time then the used duty cycle is changed to the updated duty cycle. Furthermore, the timer for periodically updating the node's power consumption is stopped and all previous buffered values for the power consumption are deleted since they are no longer valid for the new duty cycle. The timer will be started again at the start of the next active phase of the duty cycle. The cycle of turning some modules of the node on and off will be repeated until the node is turned off completely.

4.5.1 Low Power Listening

The networking framework uses the implementation of the B-MAC LPL algorithm that is standard in TinyOS 2.1 for Mica2 nodes. The sleep interval between two listening periods for the networking framework is set to $100ms$. If LPL is activated in the network, the time periods for the time slotted sending and the active period of the duty cycle need to be changed to match the new increased transmission times.

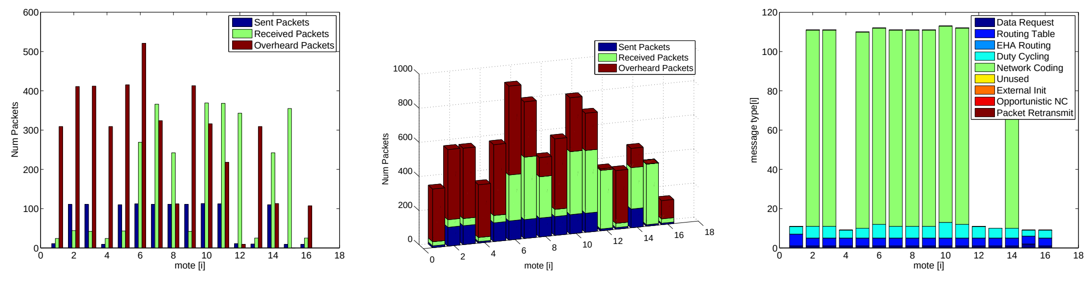
Chapter 5

Results

This section presents the results for the performance of the implemented networking algorithms and the energy management of the networking framework.

5.1 Comparison of the three Networking Types

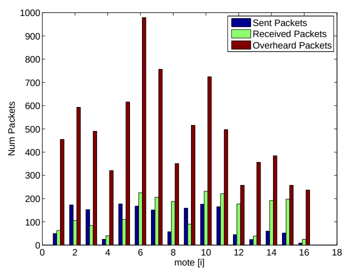
To route data from data sources to data sinks the networking framework has implemented three different networking algorithms, NC, EHAR and ONC that is running on top of the EHAR algorithm. This section presents a comparison of the performance of the implementation of the three networking algorithms. For all tests that are presented in this section each of the data source nodes sends out 100 data packets to its corresponding data sink.



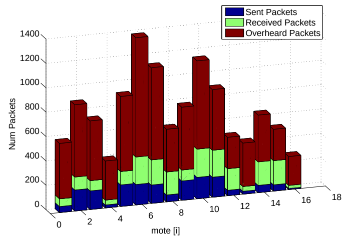
(a) Normal (b) Stacked (c) Sent packets per type

Figure 5.1: Total amount of packets sent, received and overheard per node with NC with activated duty cycling

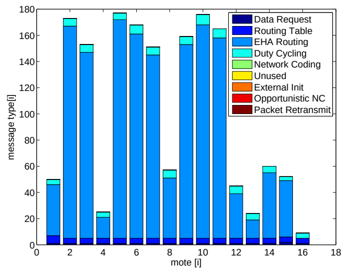
Figure 5.1 gives an overview on the packets that each of the nodes in the network is processing during one full run of the NC algorithm for a 4×4 network with activated duty cycling. Figure 5.1a and 5.1b show the total amount of packets that each node in the network is sending out, receiving and overhearing. Here it can be seen that the amount of sent out packets is pretty similar on all nodes that have to transmit packets for NC to work, while the other nodes only have to transmit some control messages. It can also be seen that the amount of overheard packets is higher than the amount of received packets. Figure 5.1c presents a more detailed view of the packets that are sent out by each of the



(a) Normal

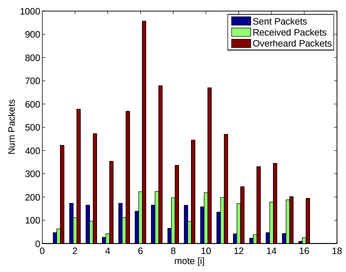


(b) Stacked

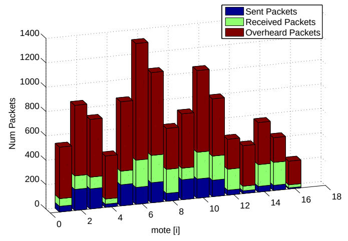


(c) Sent packets per type

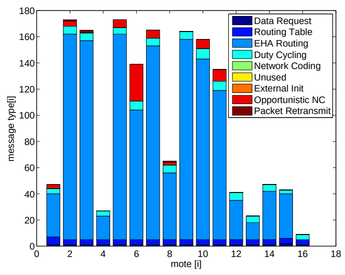
Figure 5.2: Total amount of packets sent, received and overheard per node with EHAR with activated duty cycling



(a) Normal



(b) Stacked



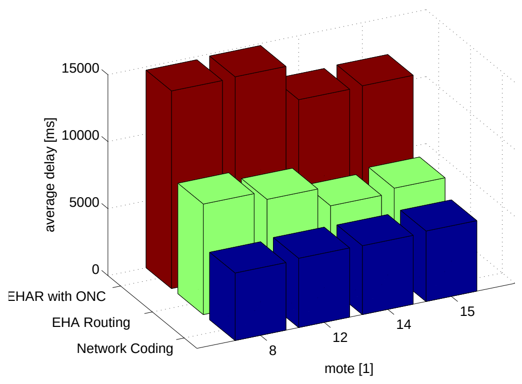
(c) Sent packets per type

Figure 5.3: Total amount of packets sent, received and overheard per node with ONC with activated duty cycling

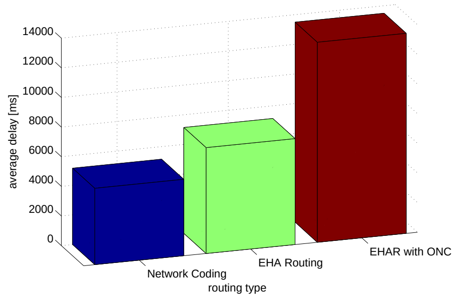
nodes in the network according to the different packet types. The total amount of NC packets that are sent out by all the nodes in the network in a 4×4 network is 1000, the amount of sent out NC packets is the same for all 10 nodes that participate in the NC algorithm. This figure also shows that the amount of packets sent out for routing table construction and duty cycling are very similar for all nodes.

Figure 5.2 presents the same overview on the packets for each node in the network for EHAR. The overview on the sent, received and overheard packets for each node in the network in Figure 5.2a and 5.2b shows that the amount of packets for all three types is larger than with NC, specially the amount of overheard packets. The detailed overview on the sent out packets per node in Figure 5.2c shows that the amount of transmitted EHAR packets is highest for the data source nodes and the nodes 6, 7, 10 and 11 that are located in the middle of the network. The total amount of sent out EHAR packets adds up to 1473 which is nearly 50% higher than the amount of packets needed by NC. Figure 5.2c also shows that except of node 16 all nodes in the network are participating in data routing.

The overview on the packets for each node in the network for ONC can be seen in Figure 5.3. The distribution of the sent, received and overheard packets with ONC which can be seen in Figure 5.3a and 5.3b is quite similar to the distribution with EHAR but the total amount of sent, received and overheard packets is lower than with EHAR. The reason for this can be seen in Figure 5.3c. With the use of ONC the amount of transmitted EHAR packets reduces to 1334 which is about 10% less than with EHAR. With the 65



(a) Average delay per destination



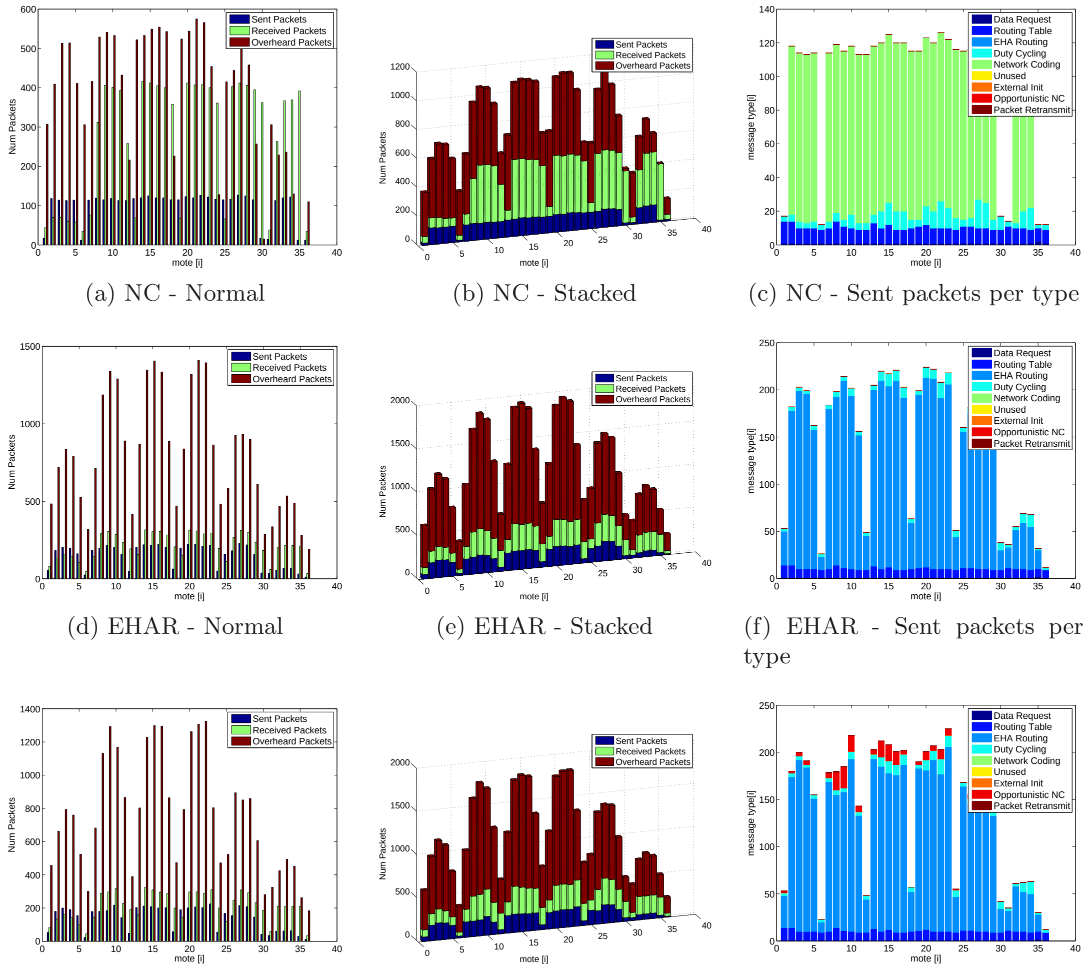
(b) Total average delay

Figure 5.4: Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling

ONC packets that are also being transmitted the total amount of needed data packet transmissions in a 4×4 network can be reduced by 5% with the use of ONC.

But the amount of sent, received and overheard packets is not the only criteria for the comparison of the networking algorithms. Figure 5.4 shows the average amount of time that passes between the generation of a data packet at a data source and the arrival of this packet at the corresponding data sink. As can be seen in Figure 5.4b with only $5000ms$ NC has the lowest average end-to-end delay while ONC needs the longest to transmit data packets to their destination with an average end-to-end delay of nearly $14000ms$. This is caused by the time that a packet needs to wait at a node to find possibilities to send out the packet combined. The average end-to-end delay of EHAR is $7000ms$. Figure 5.4a shows that the average delay of a packet also depends on the data flow since it is not the same for all destination nodes for EHAR and ONC. These deviations are caused by the time slotted sending and the chosen routes.

To compare the scalability of the presented networking types and compare their performance under different network sizes, Figure 5.5 gives an overview on the amount of sent, received and overheard packets for each node in a 6×6 network with activated duty cycling for the three presented networking types. The distribution of the amount of sent, received and overheard packets for NC that can be seen in Figure 5.5a and 5.5b looks quite similar to the one for a 4×4 network. Figure 5.5c shows that the amount of control messages that are necessary for routing table creation and duty cycling increased and that the nodes that are closer to the controller node have to transmit more control packets. This trend can also be seen in the listing of the sent packets with EHAR in Figure 5.5f and in Figure 5.5i for ONC. The total amount of NC packets needed to transmit all data to its destination nodes has increased to 3000 for a 6×6 network. The total amount of packets needed by the EHAR algorithm has increased to 4706. For ONC the amount of EHAR packets reduces to 4289 and the amount of transmitted ONC packets is 198. ONC is able to reduce the amount of EHAR packets by about 9% and the total amount of packets needed for data routing can be reduces by 5% with the use of ONC as it was also the case for a 4×4 network.



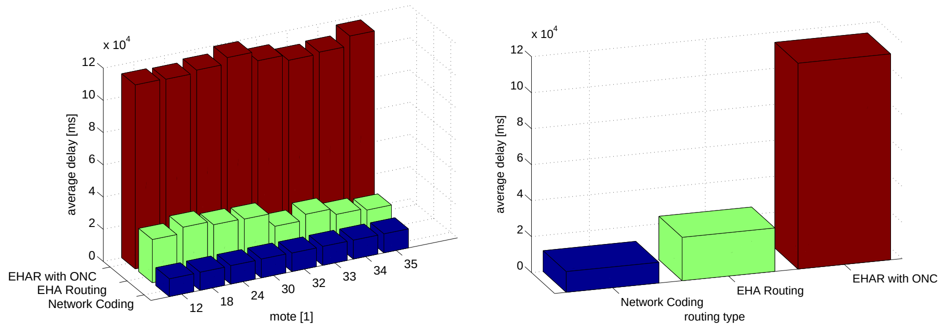
(a) NC - Normal (b) NC - Stacked (c) NC - Sent packets per type

(d) EHAR - Normal (e) EHAR - Stacked (f) EHAR - Sent packets per type

(g) ONC - Normal (h) ONC - Stacked (i) ONC - Sent packets per type

Figure 5.5: Comparison of the sent, received and overheard packets per node in a 6×6 network with activated duty cycling for NC, EHAR and ONC

Figure 5.6 shows the average time that a data packet needs from its source to its destination in a $6x$ network. Again, NC has the lowest end-to-end delay of the three networking implementations with only $10000ms$. The average end-to-end delay for EHAR increases to $25000ms$ and ONC again has the highest end-to-end delay with nearly $115000ms$. The large increase in the average end-to-end delay for ONC is also caused by an increase of the used maximum wait. In Figure 5.6a it is interesting that the average delay with EHAR is quite different between the different data sinks. The packets to the data sinks at the bottom of the network take about 5 to 10 seconds longer than the packets to the data sinks at the right side of the network. This is caused by the time slotted sending that favors data flows from the left side of the network to the right side of the network.



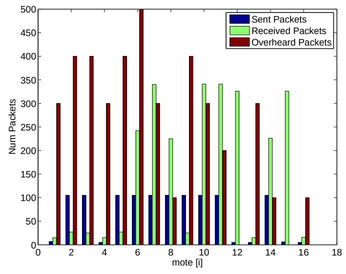
(a) Average delay per destination (b) Total average delay

Figure 5.6: Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling for a network with 6×6 nodes

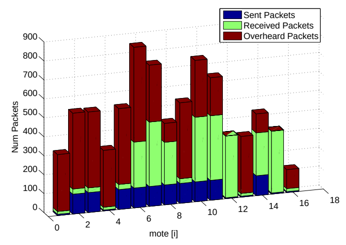
5.1.1 Comparison without Duty Cycling

To reduce the power consumption of the nodes all nodes in the network are using duty cycling. To check the influence of duty cycling on the implemented networking algorithms Figure 5.7 gives an overview on the total amount of sent, received and overheard packets in a 4×4 network with deactivated duty cycling for all three networking implementations. The only difference there is between the results that are shown in Figure 5.7 and the results for the three networking implementations with activated duty cycling is that without duty cycling there are no duty cycling control packets which reduces the total amount of packets a little bit. For the actual data routing there are no significant differences in the amount of packets. The results for EHAR and ONC can vary slightly because of the used probabilistic routing scheme.

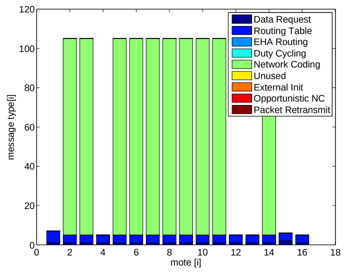
But the use of duty cycling has a great influence on the average end-to-end delay. Figure 5.8 shows the average end-to-end delay in a 4×4 network without duty cycling. Just like with activated duty cycling NC has the shortest delay and ONC the highest, the values for the average end-to-end delay are by far lower though. While the total average delay for NC with duty cycling was about $5000ms$ the total average delay without duty cycling that can be seen in Figure 5.8b is now slightly below $600ms$. The average end-to-end delay varies a bit for the different destinations as can be seen in 5.8a, for packets to node 8 the average end-to-end delay is even only $450ms$ long. This is even below the duration of one transmission period which is $480ms$ long for a 4×4 network. For EHAR the total average end-to-end delay is reduced from $7000ms$ to only slightly above $1000ms$. It is interesting to see in Figure 5.8a that the average delay for the most left bottom destination node 8 and the upper right destination node 14 is higher than the delay to the other two destinations while with duty cycling both bottom destination nodes 8 and 12 had a higher delay than the right destination nodes. For ONC the total average end-to-end delay reduces from $14000ms$ to only $2000ms$. Here the average delay is pretty even for all the destination nodes, only the most left bottom destination node 8 has a slightly higher average end-to-end delay.



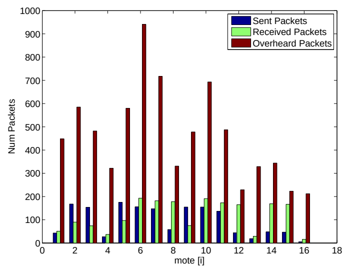
(a) NC - Normal



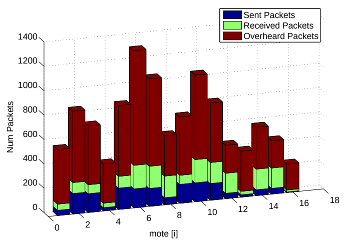
(b) NC - Stacked



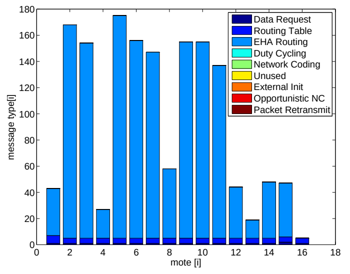
(c) NC - Sent packets per type



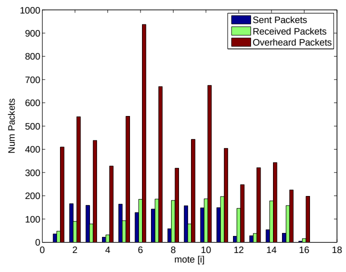
(d) EHAR - Normal



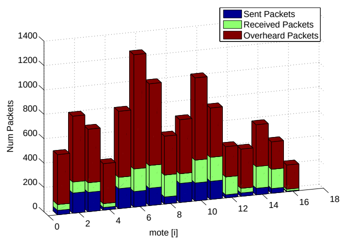
(e) EHAR - Stacked



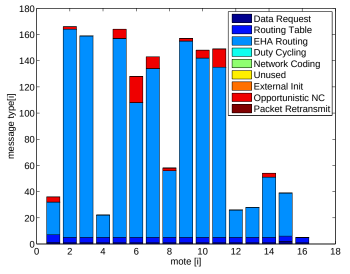
(f) EHAR - Sent packets per type



(g) ONC - Normal



(h) ONC - Stacked



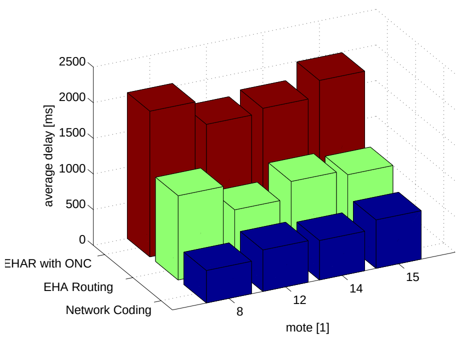
(i) ONC - Sent packets per type

Figure 5.7: Comparison of the sent, received and overheard packets per node in a 4×4 network without duty cycling for NC, EHAR and ONC

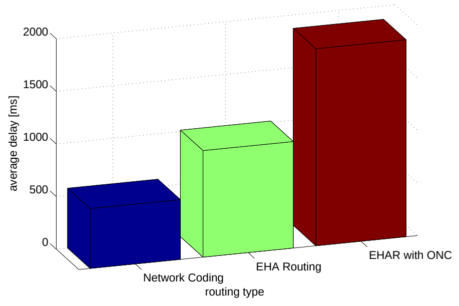
5.1.2 Measurements of data sending on intermediate node

To be able to compare the actual power consumption of three networking types, the networking framework was deployed on Mica2 sensor nodes. Due to the limited amount of available nodes the tests were performed in a $3x3$ network. The measurements that are presented in this section were all made on node 5 that is located in the center of the network.

Figure 5.9 shows the measurement for one data flow with NC where the node first receives the two packets, that it shall combine and then sends out the combined packet. The first packet is received from $9220ms$ until $9320ms$, the second packet is received from $9515ms$ until $9585ms$. The receiving of the second packet is $30ms$ shorter than the receiving of the first packet. This is caused by LPL and means that the listening for a packet transmission of node 5 occurred at a later moment of the transmission of the preamble for the second packet. From $9830ms$ until $10000ms$ the node is then sending out



(a) Average delay per destination



(b) Total average delay

Figure 5.8: Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type without duty cycling

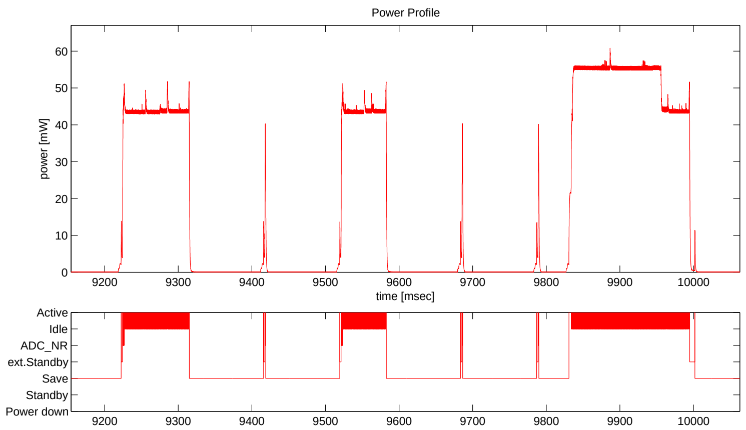


Figure 5.9: Data flow on intermediate node with NC

the combined packet again. The average amount of power that is needed during the $100ms$ of the reception of the first packets is $39.92mW$ which leads to an energy cost of $3.99mJ$ for the reception of the first packet. The average power consumption for the reception of the second packet is $38.83mW$ which leads to an energy cost of $2.72mJ$ for the $70ms$ that the reception of the second packet takes. The sending of the NC packet takes $170ms$ and has an average power consumption of $50.05mW$ which leads to an energy cost of $8.51mJ$ for sending the NC packet.

The measurement for one data flow with EHAR can be seen in Figure 5.10. Unlike with NC, the node here only receives one data packet that it then sends out again. To forward the same amount of data as was forwarded by NC in the measurement from Figure 5.9 the sequence that is shown in Figure 5.10 needs to be done two times. The node is receiving data from $22730ms$ until $22980ms$ which is far longer than the receiving times that were measured with NC. This is caused because the node is here actually also receiving two packets, but the second packet is discarded afterwards because it was only overheard. The retransmission of the received packet is then happening from $23140ms$

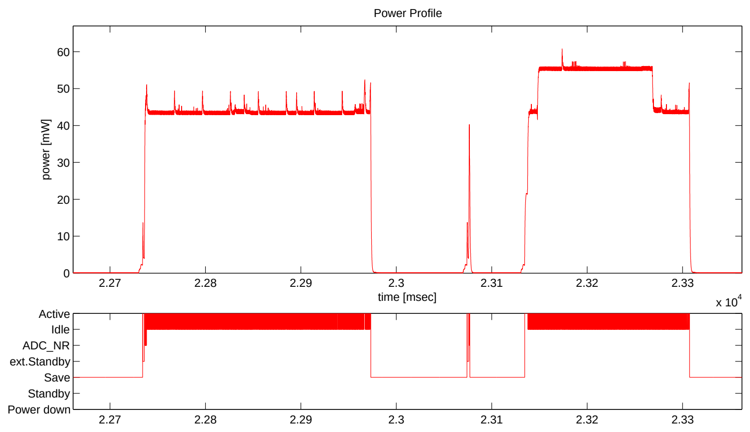


Figure 5.10: Data flow on intermediate node with EHAR

until $23310ms$ which is as long as it took with NC. The average power consumption for the $250ms$ of data receiving is $41.46mW$ which leads to an energy cost of $10.37mJ$. For the sending of the data packets the average power consumption during the $170ms$ that it takes is $51.3mW$ and the energy cost is $8.72mJ$.

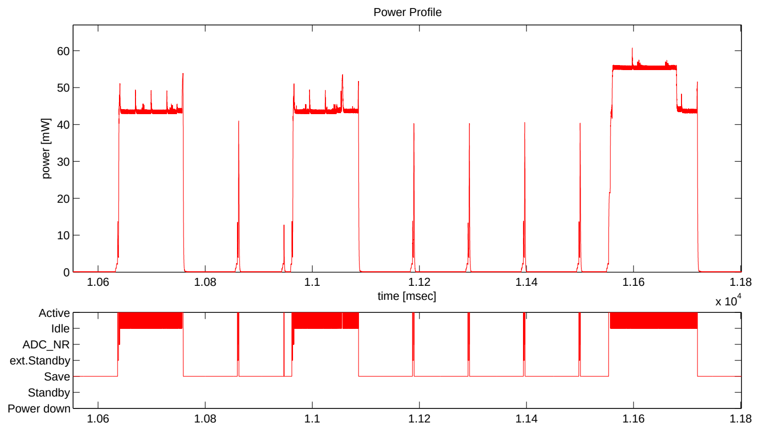
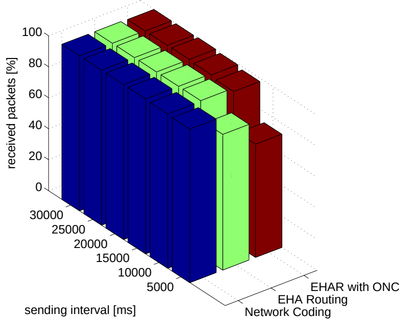


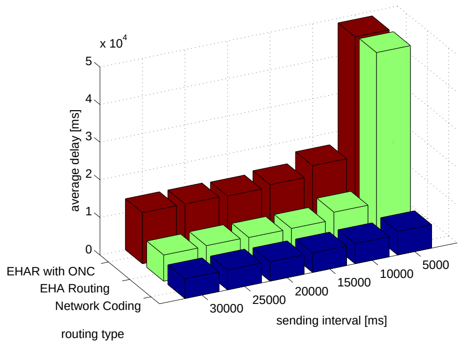
Figure 5.11: Data flow on intermediate node with ONC

The third measurement which can be seen in Figure 5.11 shows one data flow with ONC. The measured data flow shows the case where the node receives packets from two data flows that it is able to combine and thus only needs to send out one packet to forward the data. The first packet is received from $10630ms$ until $10760ms$ and the second packet is received from $10960ms$ until $11090ms$. The transmission of the combined packet is happening from $11550ms$ until $11720ms$. This shows that the actual transmitting of a data packet takes the same time for all the three networking types types. The average consumed power during the $130ms$ of the reception of the first packet is $40.69mW$ and the energy cost is $5.29mJ$. For the second received packet the duration stays the same, the consumed power changes to $41.56mW$ and the energy cost to $5.4mJ$. The sending of the ONC packet takes $170ms$ and has an average power consumption of $50.48mW$ and an energy cost of $8.58mJ$. In summary it can be seen that while the energy cost of receiving is different for the measurements, the average consumed power during receiving is pretty similar. The differences in the energy cost are caused by the different durations of receiving which is causes by LPL. If a packet starts listening to the preamble later then the duration of the receiving process will also be shorter. The duration of data sending is the same for all three networking types and also the energy cost of data sending is pretty similar for all three implementations.

5.1.3 Influence of the data sending rate



(a) Percentage received packets



(b) Total average delay

Figure 5.12: Influence of the sending rate of the user application on the amount of received packets and the total average delay for NC, EHAR and ONC in a 4×4 network with activated duty cycling

For the comparison of the different networking types the interval between data transmissions of the user application was chosen so high that it does not influence the tests. To be able to compare the performance of the three implemented networking protocols for different workloads Figure 5.12 shows the percentage of received data packets and the total average end-to-end delay for different data sending intervals. For data sending rates above $10000ms$ there are no differences on both the percentage of the received packets and the total average end-to-end delay. At a data sending rate of $10000ms$ the percentage of received packets is still the same but the delay for EHAR and ONC increases slightly. For a data sending rate of $5000ms$ the percentage of received packets for EHAR reduces to 90% and for ONC it reduces to 75% while it does not change for NC. Also the total average delay does not change for NC while it increases dramatically to nearly $50000ms$ for EHAR and ONC. This is caused by the higher amount of packets that have to be transmitted with EHAR and ONC in contrast to NC.

5.1.4 Performance under different reception strengths

In the TOSSIM simulation environment that was used for most of the results presented in this chapter the reception strength between two nodes is given in dB, a value close

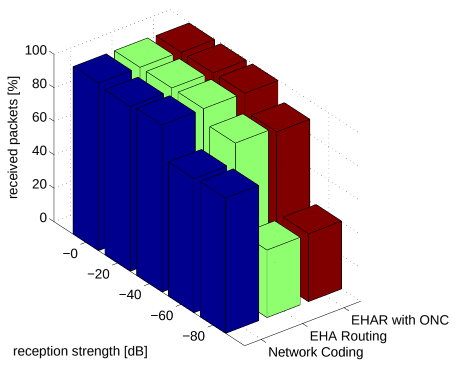
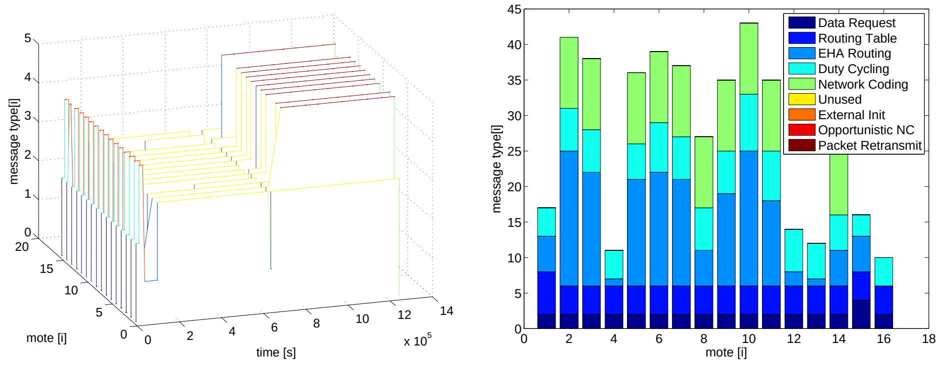


Figure 5.13: Influence of the reception strength in the network on the amount of received packets on their destination for the different networking types with activated duty cycling

to $0dB$ means that nearly all packets arrive while a value below $-100dB$ means that the nodes are not able to hear each other. Figure 5.13 compares the performance of the three implemented networking types for different reception strengths. For a reception strength of $-40dB$ or higher there is no change in the percentage of received data packets for any of the networking types. For a reception strength of $-60dB$ the percentage of successfully received packets for NC reduces to 70% while it stays at 90% for EHAR and ONC. For a reception strenght of $-80dB$ the percentage of received packets reduces significantly for EHAR and ONC, only 40% of the sent out data packets can be successfully received at the corresponding data sink. While NC is able to keep the percentage of received packets at 70%, this causes a significant amount of retransmit packets. The amount of retransmit packets only at $-80dB$ for NC is already double the amount of NC packets for a normal run. The error rate for packet transmissions at $-80dB$ is already so high though, that not even for every test run the request for starting data sending, that is broadcasted by every node in the network can be successfully received at all data sources. The worse performance of EHAR and ONC compared to NC is because the implementation of EHAR does not use acknowledgments and is thus not able to detect when a packet is lost. But it is interesting to see that ONC is able to keep its error rate at the same level as EHAR alone.

5.1.5 Changing the networking type

Since the networking framework is designed for long running WSNs it can be favorable to change the used networking type during run time. Figure 5.14 shows a short example that it is possible to change the used networking type of the framework during runtime. In this short example each of the 4 data sources in the 4×4 network sends out 20 data messages, the first 10 messages are sent with EHAR and the other 10 messages are transmitted to their data sinks with NC. Figure 5.14b shows the amount of packets that each node in the network transmitted of each packet type. Figure 5.14a shows a timeline of the transmitted packets of each node in the network, the values of the packets are corresponding to the types in Figure 5.14b, for example 3 means an EHAR packet and 5 a NC packet. The timeline shows that at about $75s$ a data request packet is broadcasted, this packet is the



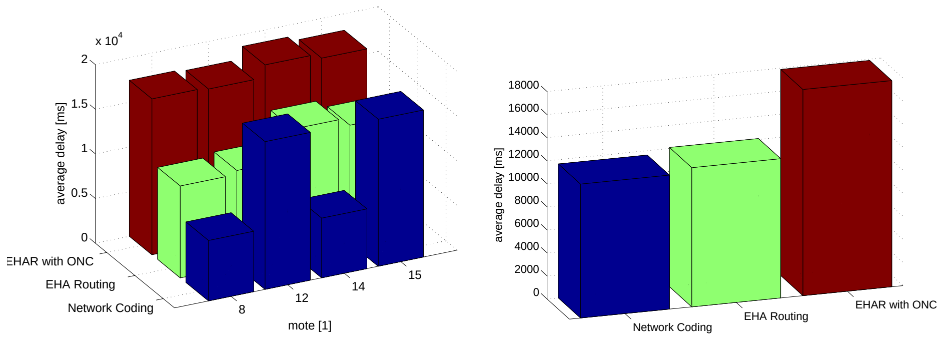
(a) Timeline of sent packets per node (b) Sent packets per type

Figure 5.14: Timeline of sent packets and total amount of sent packets per type per node for changing the networking type from EHAR to NC in a 4 × 4 network with activated duty cycling

request to change the used networking type. This packet also contains a time for when the used networking type shall be changed. Short after the broadcasting of the data request packet the networking type that is used by the nodes in the network changes to NC. The timeline always displays the last transmitted packet, this is why the nodes that do not have to send packets with NC still show that they are currently transmitting EHAR or even data request packets until the end of the simulation.

5.1.6 Influence of the sending order on the average delay

To avoid packet collisions in the network the networking framework uses time slotted sending. The order of the time slots of the different nodes in the network influences the end-to-end delay for transmitting a packet from source to destination. To test this influence the sending order in the network has been inverted. This means that the time slot that follows the time slot of node n is now the time slot of node $n-1$ and not of node $n+1$.



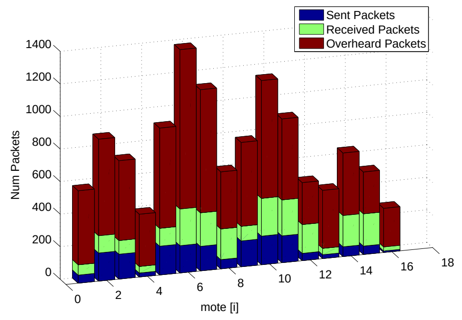
(a) Average delay per destination (b) Total average delay

Figure 5.15: Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with inverted sending order and activated duty cycling

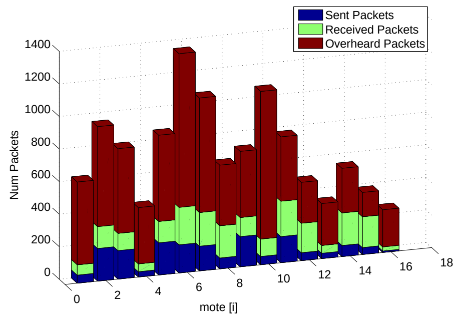
Figure 5.15 shows the time a packet needs from its source to its destination with activated duty cycling in a 4×4 network with an inverted sending order. Figure 5.15a shows that with the inverted sending order the total average end-to-end delay of a packet with EHAR and with ONC increases by about $5000ms$ and that the average delay is now pretty similar for the bottom and the right receiver nodes. The total average end-to-end delay for NC more than doubles and is now very close to the delay for EHAR with nearly $12000ms$. Figure 5.15a shows that for NC the average end-to-end delay depends highly on the position of the receiver node. While the delay on the most left bottom receiver and the top right receiver does not change much, increases the delay for the other two receivers nearly to the end-to-end delay values of ONC.

5.2 Energy Harvesting Aware Routing

5.2.1 Energy Level Adaption



(a) Same energy levels



(b) Lower energy level at node 10

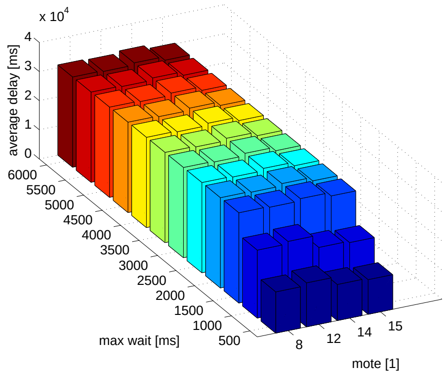
Figure 5.16: Comparison of the amount of sent, received and overheard packet for a network with the same energy level on all nodes and a network with one node with a lower energy value

The energy level of a node determines the probability that other nodes are routing their data packets via this node. Figure 5.16 compares the amount of sent, received and overheard packets per node in a 4×4 network for a setup where all nodes have the same energy level and a setup where node 10 has a lower energy level. The amount of sent and received packets at node 10 are significantly reduced with a lower energy level. The amount of overheard packets on node 10 increases by quite a lot though. This means that the sum of all sent, received and overheard packets on node 10 is only reduced by less than 10%. This means that also the power that is consumed on node 10 does not change significantly. For EHAR to be able to have an influence on the consumed power on a node the energy level needs to be lower for a region of nodes instead of just a single node. Even then EHAR can only influence the power consumption of sending, receiving and overhearing, the power consumption that is caused by listening for transmissions can not be influenced by EHAR.

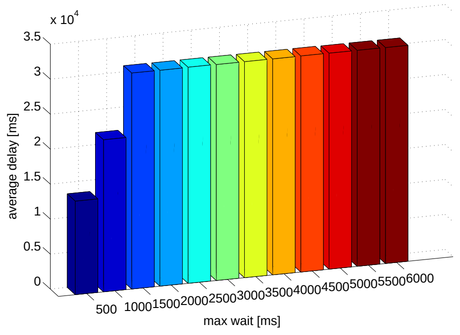
5.3 Opportunistic Network Coding

5.3.1 Influence of the Maximum Wait Time

The ONC module stores the packets that it receives from the EHAR module in a buffer to check for coding possibilities. If the module receives no other packet within the maximum wait time of the buffered packet the packet is sent out uncombined. The influence of the maximum wait time that a packet can spend in the buffer to check for coding possibilities on the average end-to-end delay of the packets in the network and the amount of packets that can be combined by the ONC module is analyzed in this section.

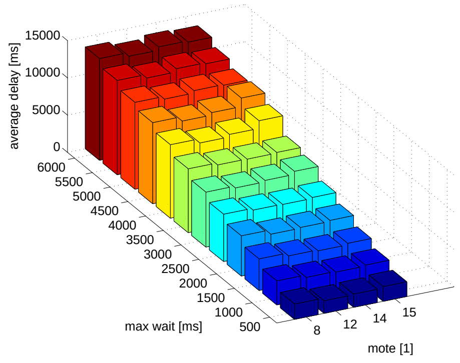


(a) Average delay per destination

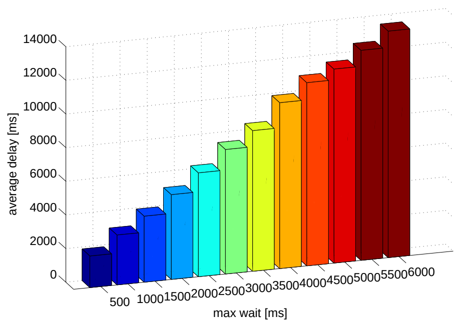


(b) Total average delay

Figure 5.17: Average delay between enqueuing a packet at the source and receiving the packet at the destination depending on the maximum wait time for ONC before transmitting a packet uncombined with activated duty cycling



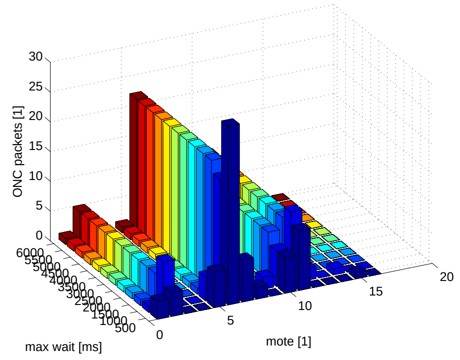
(a) Average delay per destination



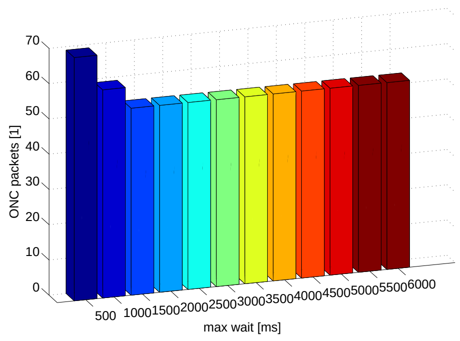
(b) Total average delay

Figure 5.18: Average delay between enqueuing a packet at the source and receiving the packet at the destination depending on the maximum wait time for ONC before transmitting a packet uncombined without duty cycling

Figure 5.17 shows the average end-to-end delay in a 4×4 network with activated duty cycling as a function of the maximum wait time of the ONC module. It is interesting to see that for a maximum wait time per node of $1500ms$ or more the average end-to-end delay does no longer change. Here the influence of the duty cycling on the average delay is higher than the influence of the maximum wait time. The duration of one active period of duty cycling is set to 4 complete transmission periods which means that the active period of a duty cycle is $1920ms$ long in a 4×4 network. This means that the maximum wait time only has an influence on the average end-to-end delay if it stays at least beneath 3/4 active period of the duty cycle. Figure 5.18 shows the same scenario with deactivated duty cycling. Here the influence of the maximum wait time on the average delay is pretty constant, an increase of the maximum wait time by $500ms$ leads to an increase of the average delay of about $1000ms$.

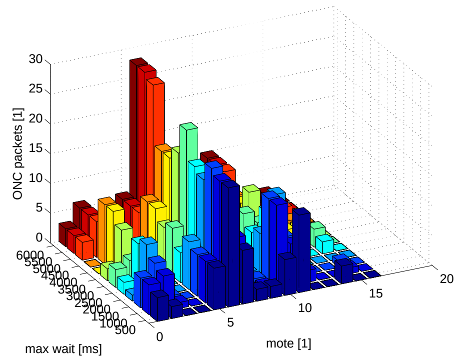


(a) Combined packets per node

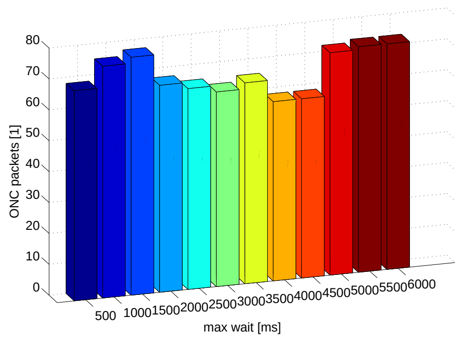


(b) Total combined packets

Figure 5.19: Amount of packets that were sent out combined by opportunistic network coding depending on the maximum wait time for ONC with activated duty cycling



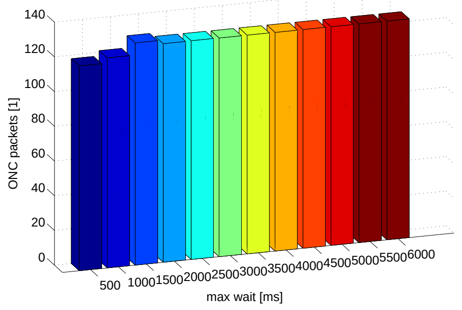
(a) Combined packets per node



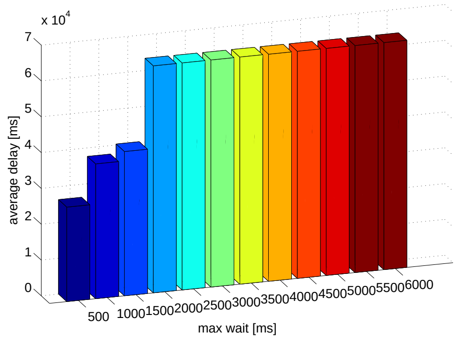
(b) Total combined packets

Figure 5.20: Amount of packets that were sent out combined by opportunistic network coding depending on the maximum wait time for ONC without duty cycling

The maximum wait time at a node does not only influence the average end-to-end delay of a data packet though, it also has an influence on the total amount of packets that can be combined by the ONC module. Figure 5.19 shows the amount of packets that are sent out combined by the ONC for one full run in a 4×4 network with activated duty cycling where each data source sends out 100 data packets. While the total amount of combined packets is the same for all maximum wait times above 1500ms the amount of combined packets varies for maximum wait times below that. The most combined packets can be achieved with a maximum wait time of only 500ms which almost matches the duration of one full transmission period which is 480ms long for a 4×4 network. Figure 5.19a shows that also the distribution of the combined packets over the network varies with the maximum wait times. Figure 5.20 shows the amount of combined packets for a 4×4 network without duty cycling. Here the maximum amount of combined packets can be achieved with a maximum wait time of 1500ms.



(a) Total combined packets



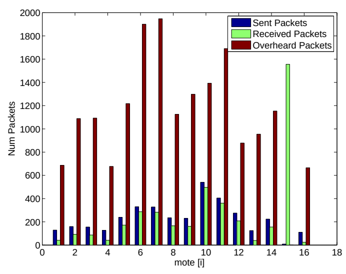
(b) Total average delay

Figure 5.21: Influence of the maximum wait time on the total amount of combined packets and average delay for ONC in a 5×5 network with activated duty cycling

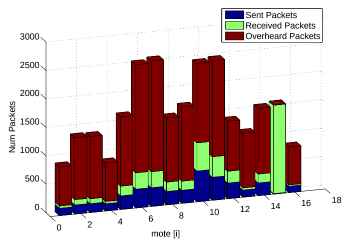
To check the influence of the network size on the optimal value for the maximum wait time Figure 5.21 shows the total amount of combined packets sent out by the ONC module and the average end-to-end delay for different maximum wait times in a 5×5 network with activated duty cycling. Figure 5.21a shows that a maximum wait time of 1500ms leads to the highest amount of combined packets. A maximum wait time of only 500ms achieves the lowest amount of combined packets for a 5×5. Since the duration of one transmission period for a 5×5 network is 750ms this shows that the maximum wait time of the ONC module should be higher than the duration of one transmission period. Just like in a 4×4 network the average end-to-end delay stays the same above a certain value of the maximum wait time, for a 5×5 the average delay does no longer change for a maximum wait time of 2000ms. The duration of one active period of duty cycling for a 5×5 network is 3000ms, this means that a maximum wait time that is above 2/3 of the duration of an active period of duty cycling no longer changes the average end-to-end delay for a 5×5 network. For a 8×8 network the optimal maximum wait time increases again, the highest amount of combined packets can be achieved with a maximum wait time of 3500ms.

5.4 Comparison of EHAR and ONC in a single data sink network

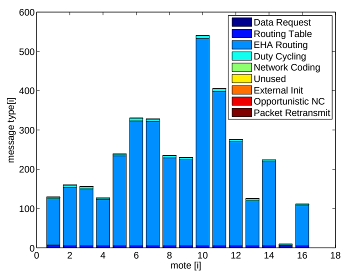
While the NC coding implementation is specifically designed for the presented mesh network layout, the implementations of EHAR and ONC are both also working in a network



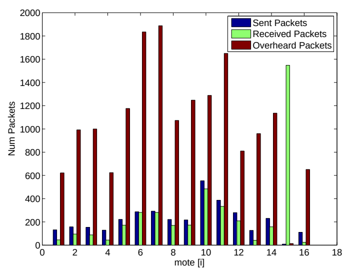
(a) EHAR - Normal



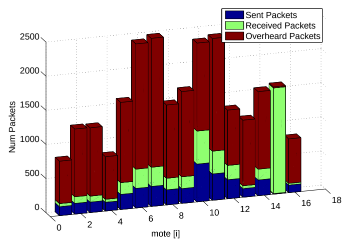
(b) EHAR - Stacked



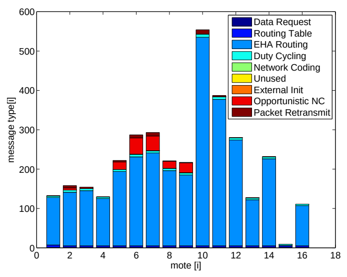
(c) EHAR - Sent packets per type



(d) ONC - Normal



(e) ONC - Stacked

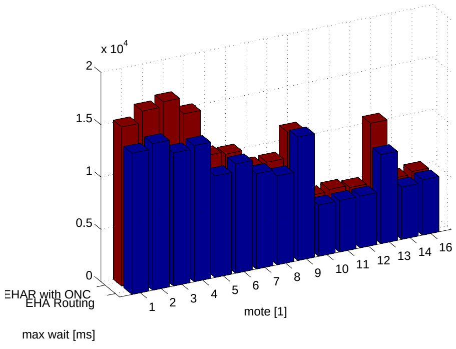


(f) ONC - Sent packets per type

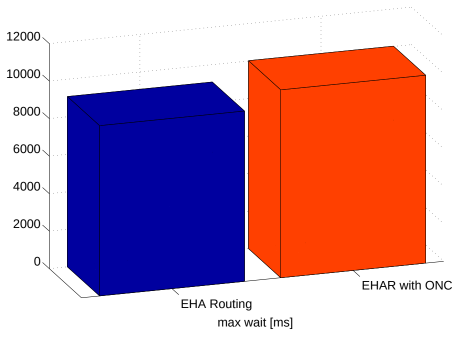
Figure 5.22: Total amount of packets sent, received and overheard per node with ONC with activated duty cycling in a network with a single data sink

with a single data sink. To be able to compare the performance of the two networking types in a single data sink network, the most right bottom receiver node acts as data sink while all other nodes in the network periodically send data to this node. Figure 5.22 gives an overview on the total amount of packets that were sent, received and overheard by each node in the network for both networking types. With EHAR the amount of transmitted EHAR packets is 3450. ONC is able to reduce the amount of EHAR packets to 3136 with an additional 158 ONC packets and 29 packets for retransmitting. Similar to the results for crossing data flows ONC is able to reduce the amount of packets but only by about 4%. The amount of sent out EHAR packets is reduced by 9% with the use of ONC. Without counting packets from the 5 data sources that can transmit their data packets directly to the data sinks which do not allow the use of ONC the percentage of combined packets increases to 4.5%. The optimum value for the maximum wait time for ONC is the same for a network with a single data sink as it is for a mesh network. One little drawback is there for the use of ONC though, because of the high amount of data flows not all packets can be recovered correctly because the node no longer has the correct packet for restoring in its overhearing queue. While some packets can be recovered after the retransmit of the necessary packet, using ONC adds an error rate of about 1% for a 4×4 network.

Figure 5.23 shows the average end-to-end delay for EHAR and ONC in a 4×4 network with a single data sink with activated duty cycling. While the difference in the end-to-end delay for crossing data flows as it can be seen in Figure 5.4 was about $5000ms$ for a network setup with only a single data sink the total average end-to-end for ONC is with $10000ms$ only $1000ms$ higher than with EHAR. For data sources that are within one hop of the data sink the average end-to-end delay is even the same as with EHAR. This is



(a) Average delay per source



(b) Total average delay

Figure 5.23: Average delay between enqueuing a packet on a source node and the packet arriving at its destination depending on the used networking type with activated duty cycling in a network with a single data sink

because these packets are immediately forwarded to the send module by the ONC module because they can not be combined with other packets.

To evaluate the possible performance of ONC in larger single data sink networks the tests have also been made for 6×6 and 8×8 networks. For a 6×6 network EHAR transmits 13147 packets to forward all data to its destination while ONC is able to reduce the amount of EHAR packets to 10999. The amount of ONC is 860 and the amount of sent retransmit packets is 457. ONC is able to reduce the amount of EHAR packets by 16.5% and the total amount of transmitted packets by 6.5%. But the percentage of successfully received packets at their destination reduces by 3% with ONC. In a 8×8 network EHAR transmits 29038 packets while ONC is able to reduce the amount of EHAR packets to 23962 for an additional 1955 ONC packets and 1193 retransmit packets. With ONC the amount of EHAR packets is reduced by 17.5% while the total amount of transmitted packet is reduces by 6.5%. Also for a 8×8 network the percentage of successfully received packets reduces by 3% with ONC. Additionally it is to note that due to the increased amount of needed packet transmissions coupled with the increased duration of a full transmission period the networking framework is not able to successfully transmit all data packets to their destination in networks with a single data sink for network sizes of 6×6 or larger. In a 6×6 network only about 90% of the packets can be received successfully and in a 8×8 network even only 60% of the sent out data packets arrive at their destination.

5.5 Duty Cycling and Low Power Listening

The networking framework has implemented a duty cycling scheme that periodically turns off the radio module to reduce the energy consumption of the nodes. Furthermore, the framework uses the already existing implementation of B-MAC in TinyOS 2 to reduce the power consumption of the radio module when it is active. This section presents measurements to evaluate the power savings that can be achieved by the use of duty cycling and LPL.

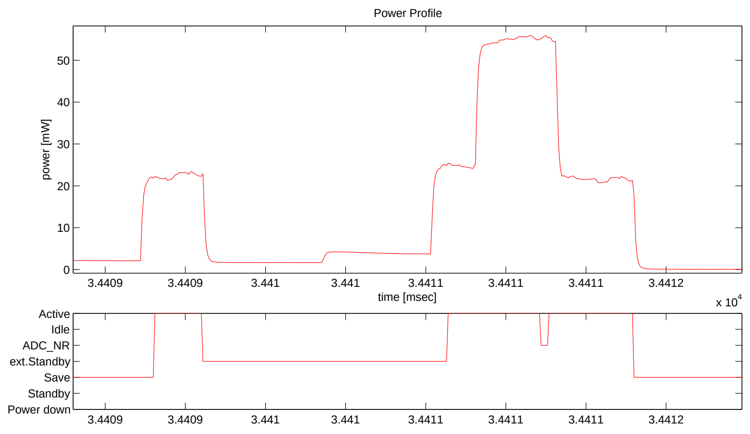
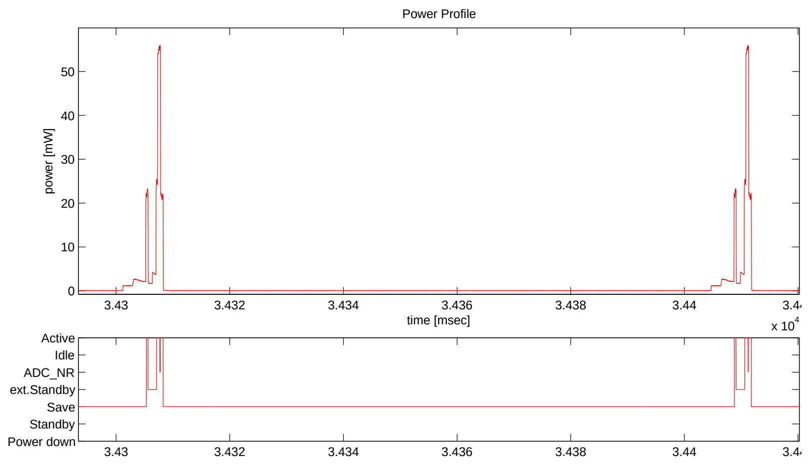


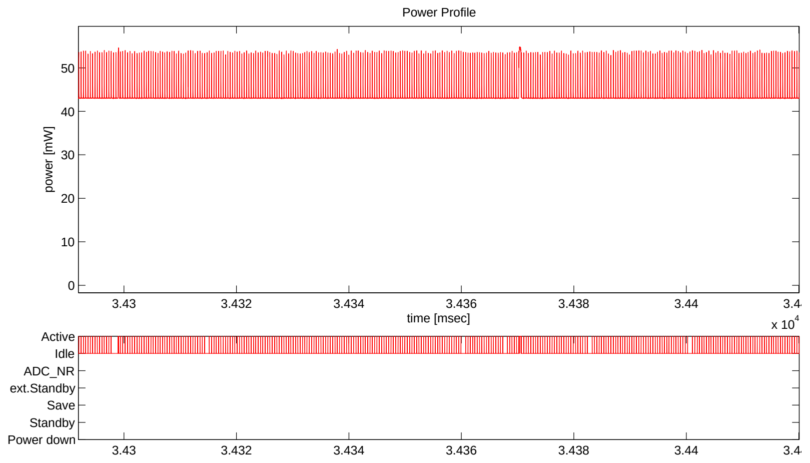
Figure 5.24: Measurement of one listening process with LPL

With activated LPL the radio module only periodically wakes up to check if there is a data transmission. Figure 5.24 shows the measurement of one listening process with LPL. The measurement shows that the listening process has two power peaks, a shorter and lower peak first and then $1.3ms$ later a second higher peak where the radio module actually checks if there is currently a packet transmission going on. The full duration of one listening process takes $3.1ms$. The first peak is $0.5ms$ long and the consumed power during this peak is $23mW$. The second peak takes in total $1.3ms$ and has three different power consumption levels. For the first $0.3ms$ the consumed power is $25mW$, then the power consumption raises to $55mW$ for the next $0.6ms$ before it decreases to $22mW$ again for the last $0.4ms$ of the second peak. The average power consumption for the whole $3.1ms$ is $18.48mW$ which leads to an energy cost of one listening process of $57.29\mu J$. Figure 5.24 also shows the power states that the processor is in during the different phases of the listening process. Before the first peak, the processor is in save mode and then switches to active for the first peak. Between the two peaks the processor goes into extended standby and then for the second peak the processor goes into active mode again with a short break inbetween where the mode changes to ADC_NR. After the second peak the processor returns back into save mode again.

Figure 5.25a shows the measurement of one full listening period for a LPL sleep interval of $100ms$. The actual time interval between two listening peaks is with a duration of $102ms$ slightly higher than the sleep interval. Between the two listening periods the power consumption of the node comes only to $0.1mW$. To be able compare the cost of one listening period of LPL to the cost of listening for transmissions without LPL Figure 5.25b shows the power consumption of a sensor node without LPL for the same duration as in Figure 5.25a. Here the power consumption is constantly at $43mW$ and every $9ms$ the radio modules checks for incoming packets for $1ms$ which increases the power consumption to $54mW$. Furthermore, it can be seen that the processor only switches between active and idle and never is able to switch into a lower mode. The average power consumption during one listening period with LPL is $0.71mW$ which leads to an energy cost of $72.42\mu J$. Without LPL the average power consumption in the same period of $102ms$ increases to



(a) With LPL



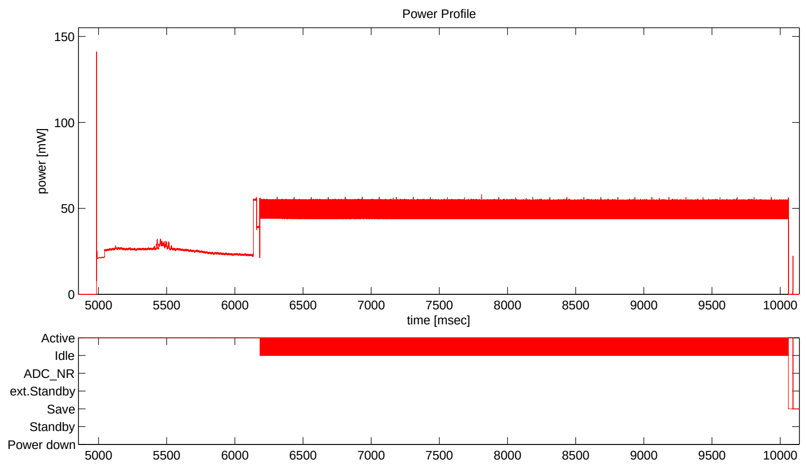
(b) Without LPL

Figure 5.25: Comparison of one LPL listening interval to the same time listening without LPL

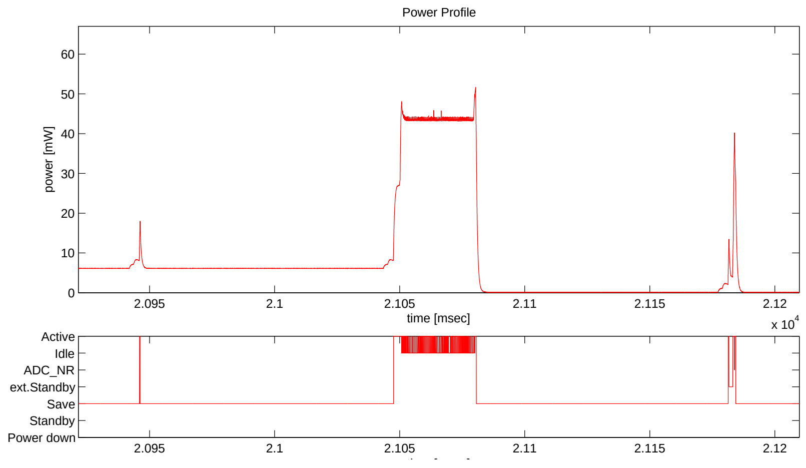
$43.99mW$ which leads to an energy cost of $4.49mJ$ which is more than 60 times higher than with LPL.

There are some additional costs for using LPL and duty cycling though which can be seen in Figure 5.26. When LPL is activated and the radio module is turned on for the first time the radio module stays active for the first $3925ms$ as can be seen in Figure 5.26a before it can start to only periodically check for packet transmissions. This is additional to the $1150ms$ that the node needs each time it is turned on after it was completely shut down to start itself and all its modules. While the power consumption during the turn on of the node is at only $25.17mW$ does it rise to $45.05mW$ while the radio module is calibrated. This means that the energy cost of the first part is $28.95mJ$ and the energy cost of the second part is $176.82mJ$ which leads to a total energy cost of $205.77mJ$ for turning a node on.

Each time the duty cycling module turns on the radio module there is also an additional cost involved. As can be seen in Figure 5.26b when the radio module is turned on the



(a) LPL turn node on



(b) DC turn on

Figure 5.26: Initial costs for activating LPL and switching to active in the DC module

power consumption increases to $43.5mW$ for $30ms$ with a $1ms$ long peak of $52mW$ at the end. The energy cost for this $30ms$ is $1.3mJ$. The higher energy level before the duty cycle is turned on is caused by a turned on LED on the measured sensor node which was used to indicate the duty cycling state. The turn on cost for duty cycling means that duty cycling only is beneficial if the power that is saved not having to periodically check for packets is higher than the cost for switching on the radio module again.

Chapter 6

Conclusion

6.1 Summary

This thesis presented the design and implementation of a networking framework specifically designed for energy harvesting powered WSNs. The implementation of the framework consists of three networking algorithms that offer different solutions to the problems of energy harvesting WSNs. Furthermore, the proposed networking framework uses duty cycling and LPL to further reduce the power consumption of the nodes in the network. While the presented framework is specifically designed for a mesh network with crossing data flows most of the framework can be adapted to any type of network.

The three networking algorithms that are presented in this thesis and implemented in the networking framework are energy harvesting aware routing, network coding and opportunistic network coding. The presented NC algorithm is specifically designed for the presented mesh network setup. While it offers the best performance in regards to number of packet transmissions and delay, its performance decreases heavily if conditions of the networking framework are changed. Just inverting the sending order of the used time slotted sending already more than doubles the end-to-end delay in a $4x4$ network. While EHAR has the highest amount of necessary packet transmissions it offers good utility and adaptability and does not suffer from changing parameters of the underlying network. ONC which is operating on top of EHAR is able to keep the flexibility of the used routing algorithm while trading an increased latency for a reduced amount of necessary packet transmissions. While the decrease in transmissions is still far from the possibilities offered by NC, the remaining flexibility proves it to very useful for many types of network setups.

But routing alone is not sufficient to reduce the power consumption of the nodes to operate energy neutral. To further reduce the power consumption the networking framework implements duty cycling alongside the already existing implementation of LPL. While LPL already greatly reduces the power consumption of the radio module, additionally duty cycling the radio module allows to further reduce the power consumption of the radio module for low power sensing networks.

6.2 Future Work

While the networking framework that is presented in this thesis is able to fulfill the requirements of energy harvesting powered WSNs, there are still topics left that offer room for improvement.

Energy Harvesting Aware Routing

The current implementation of EHAR only takes the energy level of the next hop into consideration for determining the next hop of a packet. To optimize the data paths the routing algorithm could be enhanced to chose next hops according to the energy levels of the total path.

Even if low energy nodes have to transmit less data packets, they still suffer from the energy cost of overhearing transmissions by their neighboring nodes. Thus the EHAR algorithm could be enhanced to integrate the energy level of the neighboring nodes into the probability of a possible next hop.

Network Coding

While NC offers great potential it is also very dependent on an optimized setup. Future work on NC could implement a hybrid NC and routing algorithm that is able to detect areas that allow the use of NC during runtime.

Opportunistic Network Coding

The proposed ONC algorithm is specifically designed for location-aware low power WSNs. Future work on ONC could adapt the proposed algorithm for networks where location information of the nodes is not available. For networks with higher amounts of data transmissions ONC could be extended to be able to combine more than two packets. While the proposed algorithm also works in single data sink networks an optimized implementation could increase the possible gains and reduce the increased error rate of ONC in single data sink networks.

Duty Cycling

Currently the duty cycle of the network depends on the node with the worst energy profile. Future work could implement a multi-level duty cycling that allows to increase the performance for high power nodes while keeping the necessary overhead small.

Appendix A

List of Abbrevations

BIP	Broadcast Incremental Power
CCA	Clear Channel Assessment
CHESS	Communication Using Hybrid Energy Storage System
CMMBCR	Conditional Max-Min Battery Capacity Routing
CTS	Clear To Send
DCAR	Distributed Coding-Aware Routing
EHAR	Energy Harvesting Aware Routing
EHD	Energy Harvesting Device
EWMA	Exponentially Weighted Moving-Average
E-WME	Energy-opportunistic Weighted Minimum Energy
GAF	Geographic Adaptive Fidelity
HESS	Hybrid Energy Storage System
IP	Internet Protocol
LAPAR	Location-Aided Power-Aware Routing
LNCS	Location-aware Network Coding Security
LPL	Low Power Listening
MAC	Medium Access Control
MDR	Minimum Drain Rate
MECN	Minimum-Energy Communication Network
MiLAN	Middleware Linking Applications and Networks
MIP	Multicast Incremental Power
MSR	Maximum Survivability Routing
NC	Network Coding
OMFR	Optimum Maxflow Routing
ONC	Opportunistic Network Coding
PARO	Power-Aware Routing Optimization
QoS	Quality of Service
RATS	Rate Adaptive Time Synchronization
RTS	Ready To Send
SINA	Sensor Information Network Architecture
SMECN	Small Minimum-Energy Communication Network
WSN	Wireless sensor network

Bibliography

[1] Installing TinyOS 2.1. `http://docs.tinyos.net/index.php/Installing_TinyOS_2.1`, 2010.

[2] Rudolf Ahlswede, Ning Cai, and Shuo-Yen Robert Li. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[3] Jamal N. Al-Karaki and Ahmed E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications Magazine*, 11(6):6–28, 2004.

[4] Erman Ayday, Farshid Delgosha, and Faramarz Fekri. Location-aware security services for wireless sensor networks using network coding. In *INFOCOM 2007: 26th IEEE International Conference on Computer Communications*, pages 1226 – 1234, 2007.

[5] Alessandro Bogliolo, Emanuele Lattanzi, and Andrea Acquaviva. Energetic sustainability of environmentally powered wireless sensor networks. In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 149–152, 2006.

[6] Zack Butler, Peter Corke, Ron Peterson, and Daniela Rus. Networked cows: Virtual fences for controlling cows. *WAMES*, 2004.

[7] Jae-Hwan Chang and Leandros Tassiulas. Energy conserving routing in wireless adhoc networks. In *INFOCOM 2000: 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 22–31, 2000.

[8] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, Sept 2004.

[9] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, December 1997.

[10] Prabal K. Dutta and David E. Culler. System software techniques for low-power operation in wireless sensor networks. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 925–932, 2005.

[11] Christina Fraguli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, 2006.

[12] Shashidhar Rao Gandham, Milind Dawande, Ravi Prakash, and S. Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *GLOBECOM '03: Global Telecommunications Conference*, pages 377–381, 2003.

[13] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 130–141, 2005.

[14] Philipp M. Glatz, Johannes Loinig, Christian Steger, and Reinhold Weiss. A first step towards energy management for network coding in wireless sensor networks. In *MICC 2009: IEEE 9th Malaysia International Conference on Communications*, pages 905–910, 2009.

[15] Javier Gomez, Andrew T. Campbell, Mahmoud Naghshineh, and Chatschik Bisdikian. Conserving transmission power in wireless ad hoc networks. In *9th International Conference on Network Protocols*, pages 24–34, Nov. 2001.

[16] Zheng Guo, Peng Xie, Jun-Hong Cui, and Bing Wang. On applying network coding to underwater sensor networks. In *WUWNet '06: Proceedings of the 1st ACM international workshop on Underwater networks*, pages 109–112, 2006.

[17] Salem Hadim and Nader Mohamed. Middleware for wireless sensor networks: A survey. In *Comsware 2006: 1st International Conference on Communication System Software and Middleware*, pages 1–7, 2006.

[18] Salem Hadim and Nader Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1, 2006.

[19] G. P. Halkes and K. G. Langendoen. Crankshaft: An energy-efficient MAC-Protocol for dense wireless sensor networks. In *Wireless Sensor Networks*, volume 4373 of *Lecture Notes in Computer Science*, pages 228–244, 2007.

[20] David Hasenfratz, Andreas Meier, Clemens Moser, Jian-Jia Chen, and Lothar Thiele. Analysis, comparison, and optimization of routing protocols for energy harvesting wireless sensor networks. *SUTC '10: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 19–26, 2010.

[21] Wendi B. Heinzelman, Amy L. Murphy, H.S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, Jan/Feb 2004.

[22] Tracey Ho, Muriel Médard, Jun Shi, Michelle Effros, and David R. Karger. On randomized network coding. *Proceedings of the 41st Annual Allerton Conference on Communication Control and Computing*, 2003.

[23] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani B. Srivastava, and Vijay Raghunathan. Adaptive duty cycling for energy harvesting systems. In *ISLPED '06: Proceedings of the international symposium on Low power electronics and design*, pages 180–185, 2006.

[24] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[25] Mikkel Koefoed Jakobsen, Jan Madsen, and Michael R. Hansen. Dehar: A distributed energy harvesting aware routing algorithm for ad-hoc multi-hop wireless sensor networks. In *WoWMoM 2010: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–9, 2010.

[26] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGOPS Operating Systems Review*, 36(5):96–107, 2002.

[27] Aravind Kailas, Mary Ann Ingram, and Ying Zhang. A novel routing metric for environmentally-powered sensors with hybrid energy storage systems. *Wireless VITAE Conference*, 2009.

[28] Amal Kansal, Jason Hsu, Mani B. Srivastava, and Vijay Raghunathan. Harvesting aware power management for sensor networks. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 651–656, 2006.

[29] Amal Kansal and Mani B. Srivastava. An environmental energy harvesting framework for sensor networks. In *ISLPED '03: Proceedings of the international symposium on Low power electronics and design*, pages 481–486, 2003.

[30] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(4):32, 2007.

[31] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, 2008.

[32] Dongkyun Kim, J.J. Garcia-Luna-Aceves, Katia Obraczka, Juan-Carlos Cano, and Pietro Manzoni. Power-aware routing based on the energy drain rate for mobile ad hoc networks. In *11th International Conference on Computer Communications and Networks*, pages 565–569, Oct. 2002.

[33] Lorenz Cuno Klopfenstein, Emanuele Lattanzi, and Alessandro Bogliolo. Implementing energetically sustainable routing algorithms for autonomous wsns. In *WoWMoM 2007: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, 2007.

[34] Hyuntaek Kwon, Donggeon Noh, Junu Kim, Joonho Lee, Dongeun Lee, and Heonshik Shin. Low-latency routing for energy-harvesting sensor networks. In *Ubiquitous Intelligence and Computing*, volume 4611 of *Lecture Notes in Computer Science*, pages 422–433. Springer Berlin / Heidelberg, 2007.

[35] Emanuele Lattanzi, Edoardo Regini, Andrea Acquaviva, and Alessandro Bogliolo. Energetic sustainability of routing algorithms for energy-harvesting wireless sensor networks. *Computer Communications*, 30(14-15):2976–2986, 2007.

[36] Jilin Le, John Chi Shing Lui, and Dah Ming Chiu. Dcar: Distributed coding-aware routing in wireless networks. In *ICDCS '08: Proceedings of the 28th International Conference on Distributed Computing Systems*, pages 462–469, 2008.

[37] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Operating Systems Review*, 36(5):85–95, 2002.

[38] Jiageng Li, David Cordes, and Jingyuan Zhang. Power-aware routing protocols in ad hoc wireless networks. *IEEE Wireless Communications*, 12(6):69–81, 2005.

[39] Li Li and Joseph Y. Halpern. Minimum-energy mobile wireless networks revisited. In *ICC 2001: IEEE International Conference on Communications*, volume 1, pages 278–283, Jun 2001.

[40] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[41] Weifa Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 112–122, 2002.

[42] Longbi Lin, Ness B. Shroff, and R. Srikant. Asymptotically optimal power-aware routing for multihop wireless networks with renewable energy sources. *IEEE/ACM Transactions on Networking*, 15(5):1021–1034, 2007.

[43] Ting Liu and Margaret Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *PPoPP '03: Proceedings of the 9th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, 2003.

[44] Jun Luo and Jean-Pierre Hubaux. Joint mobility and routing for lifetime elongation in wireless sensor networks. In *INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1735–1746, 2005.

[45] Jun Luo, Jacques Panchard, Micha? Pirkowski, Matthias Grossglauser, and Jean-Pierre Hubaux. Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In *Distributed Computing in Sensor Systems*, volume 4026 of *Lecture Notes in Computer Science*, pages 480–497. 2006.

[46] Vladimir Marbukh and Madhavi W. Subbarao. Framework for maximum survivability routing for a manet. In *MILCOM 2000: 21st Century Military Communications Conference Proceedings*, volume 1, pages 282–286, 2000.

[47] Stevan Marinkovic and Emanuel Popovici. Network coding for efficient error recovery in wireless sensor networks for medical applications. In *1st International Conference on Emerging Network Intelligence*, pages 15–20, 2009.

[48] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 255–265, 2000.

[49] Florian Michahelles, Peter Matter, Albrecht Schmidt, and Bernt Schiele. Applying wearable sensors to avalanche rescue. *Computers & Graphics*, 27(6):839 – 847, 2003.

[50] Clemens Moser, Lothar Thiele, Davide Brunelli, and Luca Benini. Adaptive power management in energy harvesting systems. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 773–778, 2007.

[51] Stephen Mueller, Rose P. Tsang, and Dipak Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. In *Performance Tools and Applications to Networked Systems*, volume 2965 of *Lecture Notes in Computer Science*, pages 209–234. 2004.

[52] Lilia Paradis and Qi Han. A survey of fault management in wireless sensor networks. *Journal of Network and Systems Management*, 15:171–190, 2007.

[53] Daniel Platz, Dereje H. Woldegebreal, and Holger Karl. Random network coding in wireless sensor networks: Energy efficiency via cross-layer approach. In *ISSSTA '08: IEEE 10th International Symposium on Spread Spectrum Techniques and Applications*, pages 654–660, 2008.

[54] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, 2004.

[55] Vijay Raghunathan, Saurabh Ganeriwal, and Mani B. Srivastava. Emerging techniques for long lived wireless sensor networks. *IEEE Communications Magazine*, 44(4):108–114, 2006.

[56] Theodore Rappaport. *Wireless Communications: Principles and Practice.* 2001.

[57] T.S. Rappaport and L.B. Milstein. Effects of radio propagation path loss on ds-cdma cellular frequency reuse efficiency for the reverse channel. *IEEE Transactions on Vehicular Technology*, 41(3):231 –242, aug. 1992.

[58] Volkan Rodoplu and Teresa H. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1333 – 1344, 1999.

[59] Stefan Ruff. Energy harvesting networking for wireless sensor networks. Project Thesis, December 2009.

[60] Andrea Seraghiti and Alessandro Bogliolo. Self-adapating maxflow routing for autonomous wireless sensor networks. In *SENSORCOMM '08: Proceedings of the 2nd International Conference on Sensor Technologies and Applications*, pages 360–365, 2008.

[61] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *WCNC 2002: Wireless Communications and Networking Conference*, pages 350–355, 2002.

[62] Li Shan-Shan, Zhu Pei-Dong, Liao Xiang-Ke, Cheng Wei-Fang, and Peng Shao-Liang. Energy efficient multipath routing using network coding in wireless sensor networks. In *Ad-Hoc, Mobile, and Wireless Networks*, volume 4104 of *Lecture Notes in Computer Science*, pages 114–127. 2006.

[63] Chien-Chung Shen, Chavalit Srisathapornphat, and Chaiporn Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8:52–59, Aug 2001.

[64] Eduardo Souto, aes Germano Guimar Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, and Carlos Ferraz. A message-oriented middleware for sensor networks. In *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 127–134, 2004.

[65] C.-K. Toh. Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks. *IEEE Communications Magazine*, 39(6):138–147, Jun 2001.

[66] Alberto Lopez Toledo and Xiaodong Wang. Efficient multipath in sensor networks using diffusion and network coding. In *40th Annual Conference on Information Sciences and Systems*, pages 87–92, 2006.

[67] Jeffrey E. Wieselthier, Gam D. Nguyen, and Anthony Ephremides. Energy-efficient broadcast and multicast trees in wireless networks. *Mobile Networks and Applications*, 7(6):481–492, 2002.

[68] Zhiqiang Xiong, Wei Liu, Jiaqing Huang, Wenqing Cheng, and Zongkai Yang. Network coding approach: Intra-cluster information exchange in wireless sensor networks. In *Mobile Ad-hoc and Sensor Networks*, volume 4325 of *Lecture Notes in Computer Science*, pages 209–219. 2006.

[69] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 70–84, 2001.

[70] Yuan Xue and Baochun Li. A location-aided power-aware routing protocol in mobile ad hoc networks. In *GLOBECOM '01: Global Telecommunications Conference*, volume 5, pages 2837–2841, 2001.

[71] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002: 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1567–1576, 2002.