Unsynchronized Energy-Efficient Medium Access Control and Routing in Wireless Sensor Networks

Masterarbeit
der Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von:
Philipp Hurni
2007

Leiter der Arbeit:
Professor Torsten Braun

Forschungsgruppe Rechnernetze und Verteilte Systeme
Institut für Informatik und Angewandte Mathematik (IAM)

Abstract

Various energy saving MAC protocols for all kinds of wireless networks have evolved in the past decades. Reaching from the 802.11 WLAN-standard with its power saving extension, researchers have suggested energy saving MAC protocols for use in wireless ad hoc networks and MANETs, sensor networks and personal area networks. Today's energy saving wireless MAC protocols periodically switch the radio transceiver hardware between the costly operation modes receive and transmit and an energy-saving sleep mode. The majority of the existing power saving MAC approaches tries to synchronize the state changes of the nodes in the network and introduces mechanisms to let the nodes synchronously wake up at designated points of time in order to exchange pending traffic or control messages. Such synchronization however is not easy to achieve, especially over multiple hops, and the introduction of periodic control messages for global or cluster-wise synchronization is energetically costly. With low traffic, the energetic overhead for maintaining synchronization and slot coordination may exceed the energy spent for the actual data traffic. Mechanisms that renounce on synchronization schemes are likely to be more energy-efficient in low-traffic scenarios.

This thesis investigates modifications and optimizations on recently proposed fully unsynchronized power saving MAC protocols for wireless sensor networks based on asynchronous wake-up patterns, and intended for sensor networks with low traffic requirements. The thesis begins with investigations on efficient broadcast techniques and suggests a scheme that exploits the knowledge about the neighboring node's wake intervals. It continues with outlining a performance degrading overhearing effect that can occur in dense sensor network environments with increased load when applying a fixed constant period for the node's wake-up intervals. An alternative allocation scheme of the sensor node's wake-up intervals based on a linear movement function is shown to resolve this problem in a simple and cheap manner. Further investigations topic the issue of improving the traffic adaptivity of the MAC scheme in cases of traffic between multiple senders and one receiver, which are likely to occur when packets are forwarded from numerous nodes towards one or a few base stations. The discovered mechanisms and improvements are tested out and examined in a network simulator environment and a prototype implementation on a sensor hardware testbed.
The thesis concludes with the integration of an ad hoc on-demand routing protocol and experiments to achieve higher sensor network lifetime by balancing the traffic load over multiple paths.

Acknowledgements

I hereby express my gratitude to my supervisor, Prof. Dr. Torsten Braun. With his strong support and encouragement, he shaped the outcome of this thesis to a large extent. He always gave invaluable feedbacks and advices and was always there when I needed help.

I furthermore want to express my profound gratitude to my parents, Elisabeth and Ulrich Hurni, supporting me anytime, in every situation, and to any expense.

Special thanks go to Markus Anwander for his fruitful advices on ScatterWeb and the Embedded Sensor Boards, which helped me a lot getting comfortable with the platform. Very special thanks go to Dragan Milic, who recovered a prior version of this thesis when a data loss occurred at my home computer and I mistakenly deleted every backup copy.

Contents

Chapter 1

Introduction

Wireless Sensor Networks (WSNs) have gained remarkable attention in the past decade, not only in the academic community. With decreasing cost of integrated circuits and low-power transceiver hardware, many applications of WSN become feasible and realizeable at reasonable cost. Mark Weiser's prominent vision of *Ubiquitous Computing* [1] yet only gives a clue about where the near future of distributed and pervasive computing and information technology in general leads.
Wireless Sensor Networks are autonomous systems composed of numerous entities distributed over an area or at special points of interest. Purpose and goals of WSNs can differ heavily. WSNs can be applied in many fields of our economy and our daily life: as instruments for environmental monitoring, as useful tools in business and industrial process control, for health-care and medical purposes. WSNs can provide useful services and supply important data in military campaigns and peacekeeping missions, help protect areas or valuable assets from unauthorized access, give alarm in cases of emergency. Monitoring and reporting physical and environmental values, such as vibration, temperature or noise, is the main activity of a WSN. Nodes self-organize into a multi-hop network, and collaborate in sensing and forwarding gathered information towards one or a few base stations. Communication of the sensing units is achieved over a wireless channel with low-power radio transmission units. Sensor nodes should be robust for deployment in hostile environments, low in energy usage and inexpensive in production.
WSN nodes are in most cases battery operated. As many sensor networks are meant to cover areas where physical access is difficult, nodes are often meant not to be recollected and recharged - the energy resource consists in the battery it is initially equipped with, its lifetime is thus limited. The development of energy-scavenging techniques has already led to exciting results, but still there is no easy and cheap solution to the energy restriction. It is therefore of highest priority that the nodes make optimal use of the limited energy resources.

1.1 Power Saving in Wireless Sensor Networks

The main goal of all the efforts towards energy-saving and energy-efficient operation of wireless sensor networks lies in the prolongation of the network operability, so to speak the network *lifetime*. The current research addresses this goal both at the level of hardware and software.
The development of energy-efficient low-voltage integrated circuits has already made impressive progress in the past years. Dynamic frequency scaling and advanced power-management have become accepted and widespread techniques to manage power and energy consumption in both embedded systems and general purpose computing systems. Research and development have led to sophisticated and very energy-efficient computing systems in terms of energy-efficiency, yet the measures taken often remain transparent to the application programmer and the end user.

The challenging task in the design of energy-efficient communication protocols consists in finding means to use the wireless transceiver chip only in an *on demand* manner. The radio transceiver unit should overall use less power whenever it is not or only infrequently used. In WSNs, this task is of crucial importance, as the transceiver hardware very often is responsible for a major part of a node's energy consumption.
Turning off the radio interface whenever there is nothing to be sent does not solve the problem, as the connectivity with neighboring nodes would be disrupted and must somehow be kept alive. The power management of the communication interface is a task that needs to be tackled by the entire communication protocol stack. The radio transceiver units have to be switched into a low-power sleep state for a maximum amount of time, yet still maintaining connectivity to the neighboring nodes by periodically switching to the receive state and listen for incoming traffic. In case of no outgoing and incoming transmissions, nodes should keep their radio interfaces in an energy saving state for most of the time. Sophisticated mechanisms should still allow considerable service characteristics whenever transmissions have to be handled, with respect to throughput and latency.

In the ideal case, both, sender and receiver would always switch their transceivers into the respective operation modes, whenever there is traffic to be handled, magically knowing each others intentions. However, as we are dealing with two communication endpoints and asymmetric information, this goal can and will never be reached.

1.2 Problem and Goal Statement

The goal of this master thesis consists in the design and optimization of medium access control and routing mechanisms tailored for use in wireless sensor networks renouncing on costly synchronization schemes.

We lean on basic concepts introduced by *Braun et al.* in [2], [3] and on principal operation characteristics of the WiseMAC [4] protocol. In contrast to a huge amount of energy-saving MAC protocols, these propositions suggest to renounce on any kind of network-wide or clusterwise synchronization for the channel access, as it is done in many scheduled protocols, nor for the coordination of a common wake and sleep pattern. They neglect to exchange costly MAC layer control messages in the belief that for wireless sensor networks with low traffic requirements, maintaining a multi-hop synchronization scheme to control scheduled medium access is too costly. When the network load is low, the energetic overhead for maintaining coordination of slot or schedule assignment may easily exceed the energy spent for the actual data traffic.
A further advantage of unscheduled protocols is the fact that the nodes' wake intervals are naturally distributed and spread over time. The medium is not only allocated in a common periodic fixed slot window. With dense networks, a few nodes will always be awake in any point of time. This is a conceptual advantage, as the medium will be utilized more equitably over time.

We aim to develop an unsynchronized MAC and routing prototype for wireless sensor networks with lowest-possible duty cycle in case of no traffic, which is still flexible with increasing traffic. By closely coupling MAC and routing layer, we aim to make performance gains by exploiting cross-layer optimizations.
We target to evaluate the performance of the prototype implementation in wireless sensor network traffic scenarios with network simulation tools and on a real sensor hardware testbed. We aim to measure transmission characteristics of the prototype and energy consumption both in simulation and on real sensor hardware.

1.3 Contributions

The main contributions and insights of this thesis are summarized as follows:

- We designed an energy-efficient broadcasting technique for application in wireless sensor MAC protocols based on asynchronous wake intervals and the low duty cycle preamble sampling technique. The necessary prerequisites and assumptions under which this technique proves to deliver energy-efficiency gains in comparison to existing techniques are analytically evaluated and formally proved. The technique proves to deliver significantly better results in simulation and slightly better results in practice on real sensor hardware.

- We show how careful investigations on simulation parameters pay off when cross-comparing the obtained results results to measurement results on real sensor hardware. We show that the gap between simulation model and measurements on real devices can be reduced when carefully investigating on the parameters of the hardware testbed and modelling implementation specific settings. We show that obtaining a realistic model of the reality in simulation is up to a certain possible in small-scale experiments.

- We designed a novel allocation and arrangement scheme of the node's unsynchronized wake-up intervals to avert dangerous performance degrading effects of systematic overhearing. Nodes wake up in respect to a linear movement function in-between the fixed cycle. The wake-up intervals thereby remain deterministic and predictable for each neighboring node. The advantage of the scheme is proved analytically and on the experimentation testbeds.

- We suggest a scheme to improve the traffic-adaptivity extension of the WiseMAC [4] power saving MAC protocol in case of traffic between multiple senders and one receiver. The novelty consits in a so-called *stay-awake promise* - nodes receiving and forwarding packets from multiple stations *promise* to their neighbors to *stay awake* for further transmissions with a flag bit in the acknowledgement. Receiving these, nodes with pending transmissions can seamlessly continue to forward their frames, without having to wait for the next wake-up in the next slot. The technique is shown to improve throughput and end-to-end delay.

- We experiment with on-demand multipath routing approaches to achieve lifetime gains by balancing the load more equitably over the network and by exploiting cross-layer information. Slight performance gains can be claimed with altering the path update rules of existing on-demand routing schemes. Problems encountered with concurrent traffic along interferring paths are identified to be a direct consequence of the special properties of the MAC scheme.

1.4 Thesis Outline

Chapter 2 introduces into related work on power saving MAC protocols for wireless channel networks that influenced this work and the mechanisms described in it. It summarizes research work on ad hoc routing protocols and multi-path routing schemes for ad hoc networks. It concludes with recent work on data aggregation techniques in wireless sensor networks. The part gives an insight into ideas and principles that were studied and investigated on during the last couple of months to achieve the goals of this thesis.

Chapter 3 describes the two experimentation testbeds that were used to perform experiments, a simulation environment and a sensor hardware testbed.

Chapter 4 contains the first achievement of the thesis, which forms the entry point of the investigations on further improvements. It describes the implementation of the recently proposed power saving protocol WiseMAC, implemented on both the simulator and the sensor hardware testbed.

In Chapter 5, the thesis outlines advantages and drawbacks of the WiseMAC power saving MAC protocol. Inspired by recent work on unsynchronized power saving schemes, it proposes mechanisms to improve WiseMAC performance in different contexts. The thesis investigates theses mechanisms on the simulator and on the ESB platform.

Chapter 6 gives an insight into efforts that were undertaken to port the energy efficient MAC scheme of chapter 5 to an on-demand routing scheme and achieve further performance improvements by altering routing mechanisms and exploiting specific MAC layer information on the network layer.

Chapter 7 summarizes and concludes the thesis, refers to encountered problems and difficulties and suggests topics and fields for further investigations.

Chapter 2

Related Work

2.1 Medium Access Control in Wireless Sensor Networks

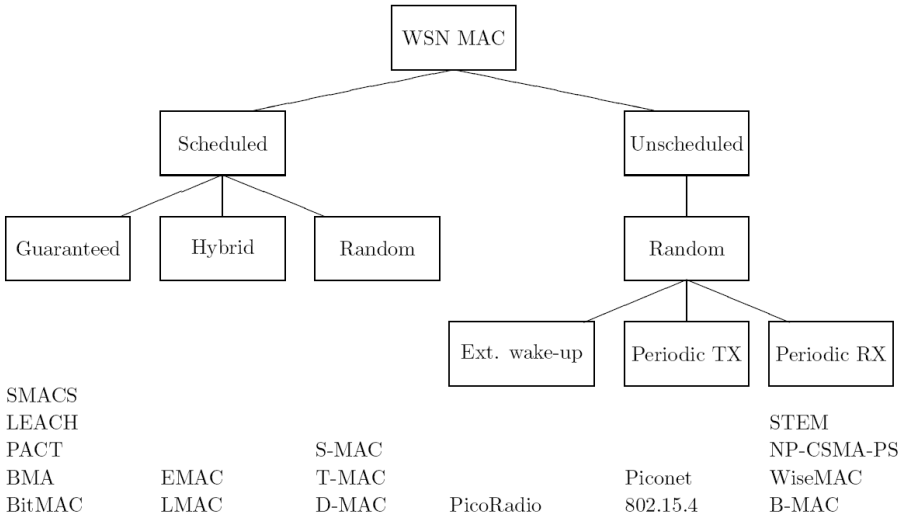


Figure 2.1: Classification of wireless sensor MAC protocols

In the past decade, various MAC mechanisms have been proposed for use in wireless sensor networks. One can find almost any wireless channel multiplexing technique (i.e. SDMA, TDMA, CDMA, FDMA) and any medium access scheme suggested for use in wireless sensing systems. The major part of the currently available sensor hardware testbeds are equipped with low-power single-channel-transceivers - so do the sensor boards that were used in the experimental part of this thesis. We therefore limit this section to related work on single-channel MAC protocols. In the following, we briefly introduce into the operation principles of the sensor MAC protocols that influenced the approach pursued in this thesis.

It is an uneasy task to organize and classify the big variety of the proposed sensor MAC protocols. In [5], *El Hoiydi* points out that all protocols share the goal of saving energy by letting nodes sleep the biggest part of the time. The main difference lies in the organization of the sensor nodes wake times. In [5], a general distinction between *scheduled* and *unscheduled* wireless sensor MAC protocols is proposed. Figure 2.1 depicts the resulting classification of the most prominent wireless sensor network MAC protocols.
In *scheduled* MAC protocols, the sensor nodes have to be synchronized. Transmissions take place during time slots that are either allocated to links, to single nodes or to groups of nodes. Slots for single nodes can either be defined for transmission or reception, be segmented to allow both directions (i.e. with windows), or be contention-based inside the slots. Problems arise when determining how the slots are assigned and by which node.

Suggestions for this problem reach from single instances that allocate the slots to the nodes, over clustered approaches with clusterheads assigning the slots to nodes in their vicinity, up to completely distributed solutions. Once the medium access is successfully split into slots and the slots assigned, *scheduled* protocols offer the major advantages of frame-based TDMA protocols, which lie in the inherent energy-efficiency due to the absence of collisions, overhearing, and idle-listening. Drawback of this approach is the high system complexity of the time slot allocation, which requires continuous periodic control message exchange to solve the allocation problem. One has to establish a tight synchronization scheme such that the node's individual clock drifts don't lead to overlapping time slots and concurrent channel access.
With *unscheduled* MAC protocols, wake-up schedules of each node's duty-cycles have to be exchanged, to make sure that nodes know when their destinations are awake when aiming to transmit a packet. Access to the medium is at random, collision avoidance often bases on contention. *Unscheduled* MAC protocols are cheaper in respect to the system maintenance overhead. Once deployed, the challenge with *unscheduled* contention-based protocols consists in finding ways to reduce the energy waste caused by collisions, overhearing, and idle-listening.

Energy efficient MAC layer techniques are a challenging and demanding issue. The research results of the various techniques that have been proposed so far suggest that there is no ultimate and sole solution to the problem. It seems that the protocol performance often varies with the underlying network topologies, the traffic pattern and traffic intensity, as well as the application scenarios.
Overall, energy efficient MAC techniques aim to operate as efficient as possible, targeting to reduce the major sources of energy waste at the MAC layer. There is a widespread agreement in the research community in the question what the sources of energy waste on the MAC layer are. The major sources or energy waste have been pointed out by *Ye et al.* in [6]:

- *Idle Listening:* If a node constantly remains in the idle state in order to wait for possibly incoming traffic, a lot of energy is wasted. For most wireless transceivers, the power consumption in the idle listening state is not much lower than in the receive state, besides some energy spent for the signal processing. For many transceivers, this small difference is barely measurable. In general, nodes should only be switched to the receive state when they actually receive frames, and switch back to sleep immediately.

- *Overhearing:* Reception of data frames that are not actually meant and destined to the receiving station represent an energy waste. Systematic overhearing and reception of other stations traffic should be avoided. Overhearing avoidance mechanisms should make sure that overhearing is improbable and infrequent.

- *Collisions:* Unrecoverable frames due to interferences with other nodes' concurrent transmissions are a waste of energy, as Automatic Repeat Request (ARQ) schemes require costly retransmissions. Introducing collision avoidance mechanisms to make collisions less probable often pays off in respect to energy consumption.

- *Overhead:* Control messages and signalling information in frame headers in order to sustain the MAC protocol operation only allow the communication scheme to operate as intended, but do not carry any useful payload. Therefore, the amount of signalling information on the MAC layer has to be kept as low as possible. If possible, signalling information should be piggybacked on data packets carrying a payload. In case of no or only sparse traffic, the control message overhead should sink to a minimum.

2.1.1 Sensor MAC

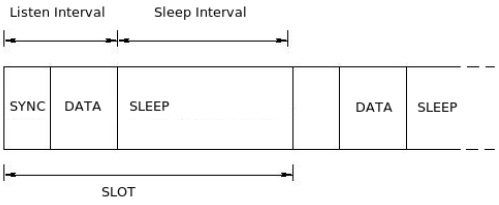


Figure 2.2: S-MAC slots

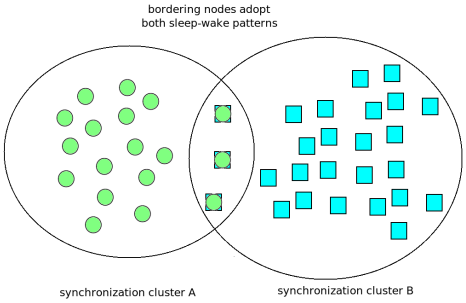


Figure 2.3: Two S-MAC clusters with bordering nodes adopting both patterns

The Sensor-MAC (S-MAC) [6] protocol is a *scheduled* protocol following the taxonomy of figure 2.1. It synchronizes the wake-up patterns of the nodes in a network, lets the nodes simultaneously wake up and fall back to sleep, but renounces on central instances. S-MAC follows a so-called virtual clustering approach to synchronize the nodes to a common wake-up scheme with a slotted structure.
Nodes regularly broadcast SYNC packets at the beginning of a slot, such that neighboring nodes receiving these packets can adjust their clocks to the latest SYNC packet and correct their clock drifts. In a bootstrapping phase, the nodes listen for incoming SYNC packets in order to join the ad-hoc network and follow to the propagated wake-up pattern of a synchronization cluster. When not hearing any SYNC's, a node starts alternating in its own wake-up pattern and propagates it with own SYNC messages, hoping that other nodes join this cluster. A problem of the virtual clustering arises when multiple clusters evolve in a multi-hop network topology, as depicted in figure 2.3. Bordering nodes between both clusters then have to adopt the wake-patterns of both clusters, in order to provide interconnecting links. By imposing twice the duty cycles on these nodes, they become vulnerable spots in the network topology, as they might drain out of energy first, whereby the network could become segmented into disjoint clusters.

An S-MAC slot consists in a listen interval and a sleep interval. The listen interval is fragmented into a small synchronization phase to exchange SYNC messages, and a second and third phase, dedicated to RTS-CTS exchange. During the second and third phase, nodes with pending traffic announce their transmission to the receivers with RTS messages. Receivers acknowledge the reception of the traffic announcement and reserve the channel by emitting a CTS. Nodes receiving a RTS will stay awake during the sleep interval, whereas all other nodes go back to sleep for the rest of the slot. Using a RTS-CTS handshake, S-MAC achieves both collision avoidance and overhearing avoidance, however at the cost of the two additional control messages.
With the S-MAC protocol, the duration of the listen interval (duty cycle) must be set in a fixed manner, which severely restrains latency and maximal throughput. The listen interval has to be long enough to handle all communication needs of the network which may arise in the worst case. If in any case more traffic has to be handled, nodes must buffer their outgoing packets until the next listening interval. There is no mechanism to prolong the current duty cycle in case of more load. This can be disadvantageous, as in sensor networks, traffic can often be of bursty nature and the rate of traffic can change over time.

2.1.2 Timeout MAC

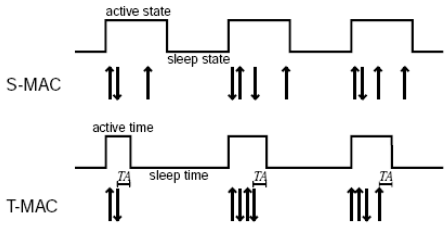


Figure 2.4: T-MAC adaptive duty cycle

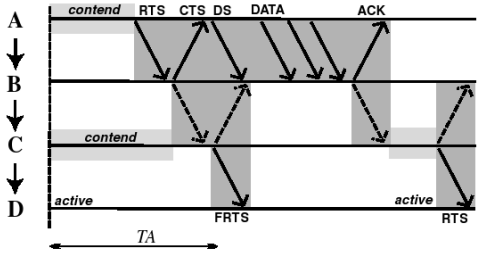


Figure 2.5: T-MAC Future Request to Send (FRTS)

The fixed duty cycle duration in S-MAC is a rather inflexible scheme. Keeping a fixed listening interval of some 10-20% of the slot duration limits the throughput and does not adapt to varying traffic. A fixed duty cycle results in significant energy wastage when traffic is low. Furthermore, packet delivery latency is very high, as every node has to wait for the next wake-up of the node in line when forwarding packets towards a sink.

The Timeout-MAC (T-MAC) [7] protocol can be considered as the successor of S-MAC and has been proposed to enhance S-MAC under variable load. It keeps the S-MAC concept of the slot windows for SYNC, RTS and CTS, as well as the common wake-up pattern concept of the virtual synchronization clusters. The problem with bordering nodes that have to adopt the wake-up patterns of two clusters therefore arises in T-MAC, too. The listening interval in T-MAC is adaptive and is extended if there are signs of pending transmissions. It ends when no so-called *activation event* has occurred for a given time threshold called *time-out* TA, as depicted in figure 2.4, which is much shorter than the fixed listen interval in SMAC. An activation event may be the sensing of any communication in the neighborhood, the end of the own data transmission or acknowledgement, the overhearing of RTS or CTS control messages which may announce further packet exchanges.

One drawback of T-MAC's adaptive time-out policy is that nodes often go to sleep too early. This problem often occurs when nodes have to forward packets over multiple hops, a typical situation is simplified in figure 2.5. The figure displays four nodes A, B, C, D forwarding traffic along the chain $A \rightarrow B \rightarrow C \rightarrow D$, where every node is only in range of the next hop. When both nodes A and C need to deliver packets to their next hops, only one transmission will succeed, as one of the destinations will go to sleep too early. Consider the following case: C looses contention to A and overhears the CTS of B. C will have to remain silent. The time-out timer of D however will run out, as D does not receive any CTS, so D goes to sleep. After the end of A's transmission to B, C will send a RTS to D to notify it about the upcoming transmission. Node D however has not heard anything for a long time and has already switched to the sleep state. So A must wait until the next T-MAC slot to try again to reach B.
To alleviate the impact of the so-called *early sleeping problem*, T-MAC introduces the *Future Request to Send (FRTS)* control message exchange depicted in figure 2.5. Shortly after receiving node B's CTS, Node C sends out a FRTS for D to keep it awake for the upcoming transmission. The scheme proved to pay off when the network load is high, but certainly leads to a higher overall system complexity. Sending out additional MAC control messages carrying no payload just to save some packet inter-delay remains questionable in sensor networks with scarce energy resources.

2.1.3 Wireless Sensor MAC

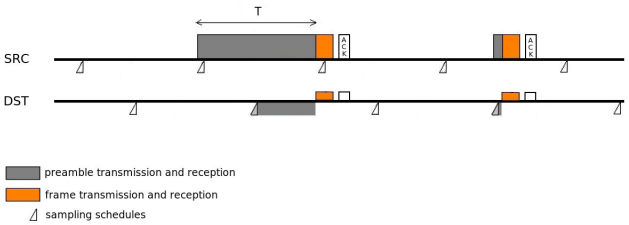


Figure 2.6: WiseMAC nodes with basic interval duration T and short medium samplings

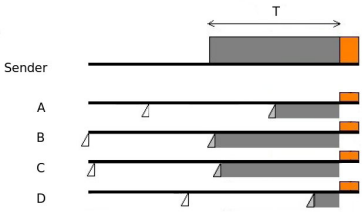


Figure 2.7: WiseMAC broadcast

The Wireless Sensor MAC (WiseMAC) [4] protocol belongs to the *unscheduled* protocols in the taxonomy of figure 2.1. WiseMAC's wake-up scheme consists of simple periodic wake-up's and duty cycles of only a few percent in order to sense the carrier for a preamble signal, as depicted in figure 2.6. All nodes in the network sample the medium with a common basic interval duration T, but their wake-up patterns are independent and left unsynchronized.
When transmitting a frame, a preamble of variable length is prepended for alerting the receiving node in its wake-up interval not to go to the sleep state. When the receiver's wake-up pattern is yet unknown, the duration of the preamble equals the full basic interval duration T, as illustrated in figure 2.6 in the first transmission. The preamble is a simple bit-sequence indicating an upcoming transmission to a node's neighborhood. The own schedule offset is then piggybacked to the frame and transmitted to the receiver. After successful frame reception, the receiver node piggybacks its own schedule to the respective frame acknowledgement. The received schedule offsets of all neighboring nodes are subsequently kept in a table and periodically updated. Based on this table, a node can determine the wake-up patterns of all its neighbors, which in turn allows to minimize the preamble length for the upcoming transmissions and thus the transmission costs. As the sender node is aware of the receiver's wake-up pattern, it only prepends a preamble that compensates for the maximun clock drift that the two involved node's clocks may have developed during the time since the last schedule exchange. As illustrated in 2.6 in the second transmission, WiseMAC minimizes the preamble and calculates its duration as follows:

$$T_{preamble} = min(4\theta L, T)$$

θ denotes the quartz oscillator clock's drift, L denotes the time since the last update of the neighbor's wake pattern and T denotes the common basic interval duration. One has to consider $4\theta L$ because the clock drift of the sender leads to an inaccuracy in the estimation by $\pm\theta$. As the receiver's drift may also have drifted apart by $\pm\theta$, one has to deal with a total inaccuracy of $\pm 2\theta$. By incorporating $4\theta L$ as minimum preamble, WiseMAC accounts for every possible drift. The length of the preamble therefore increases linearly with the time since the last schedule update, but does not exceed the duration of the common basic interval duration T. With a preamble of this size, the sender node can be sure to reach every node's medium sampling in its transmission range.

WiseMAC exploits this property to implement the broadcast technique. Broadcasting consists in prepending a preamble of the length of the basic interval duration T to every broadcast message, in order to first alert every neighboring node for the upcoming transmission, and finally transmit the frame. As illustrated in Figure 2.7, this broadcasting scheme uses a lot of energy only for sending and receiving the long preamble, whereas

the actual data transmission is very short. Broadcast is used by many multi-hop routing protocols such as DSR, AODV or data-centric approaches such as Directed Diffusion. As using such long preambles in every node is energetically costly, the designer *El Hoiydi* reflects in [5] that *more sophisticated broadcasting and flooding techniques for multi-hop ad-hoc sensor networks and MANETS remain to be designed.*

2.1.3.1 Extended Carrier Sensing Range

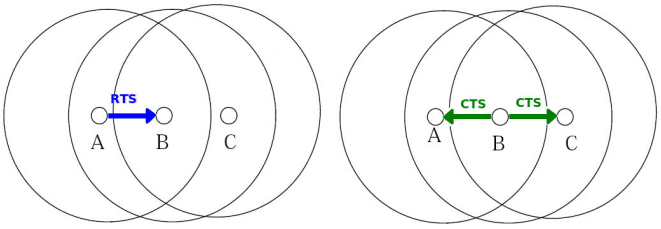


Figure 2.8: RTS-CTS handshake

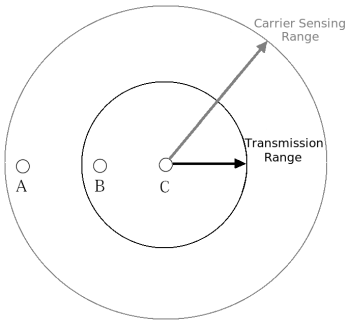


Figure 2.9: Extended carrier sensing range

In WiseMAC, the carrier sensing range is chosen to be larger than the transmission range in order to mitigate or alleviate the hidden node problem, but yet renounce on an expensive RTS-CTS control message exchange. Such a scheme would rather be difficult to achieve in WiseMAC, as the wake-up intervals of all neighboring nodes are left unsynchronized. To achieve a RTS-CTS based channel reservation, the sender would need to announce the transmission with a RTS to the receiver, which in turn would have to broadcast a CTS message to silence the surrounding nodes. However, sending a RTS and a full-preamble CTS broadcast control message to achieve a medium reservation and collision avoidance for one single message exchange would rather be inefficient.
With a so-called *extended carrier sensing range*, a node considers the medium to be busy whenever it detects a transmission or transmission noise from distant stations, even if the transmitting station is not in the node's transmission range. This more prohibitive carrier access mechanism can be achieved by setting a lower carrier sense threshold of the wireless transceiver chip's measured received signal strength indicator (RSSI). Nodes will interpret the medium to be busy, even if the received signal strength is too low to correctly demodulate frames.
The extended carrier sensing range is a sophisticated protection against the hidden node problem. All nodes located in the carrier sensing range, which is proposed to be approximately twice as large as the transmission range, will consider the medium busy and back off their own transmission attempts. However, this property is gained at the cost of a reduced overall maximum throughput, as fewer nodes will be allowed to transmit at the same time. In a sensor network designed for low-traffic scenarios, this throughput reduction can be acceptable. If resources are scarce and lifetime is more important than throughput, an extended carrier sensing range is a cheap and appropriate solution.

Figures 2.8 and 2.9 depicts both concepts to solve the hidden node problem. In both cases, node A contends to transmit a message to node B, and node C concurrently contends for transmission. Without any measures, A and C would sense the carrier idle and start transmitting their frames, as they are not in each others range. With the RTS-CTS exchange illustrated in figure 2.8, A first sends a RTS message to B, which quickly clears the channel by broadcasting a CTS message. When receiving the CTS, C stays quiet

upon overhearing the acknowledgement of B destined to A, or a NAV timer running out. In figure 2.9, A starts transmitting, while C senses the carrier and interprets it to be busy due to a lower carrier sense threshold. C may not be in range of A, but is assumed to still notice a farther away transmission by observing the RSSI values of the wireless transceiver chip.
Both concepts have been proposed and applied in various other wireless channel protocols before, problems and traffic characteristics are well-studied in the respective literature. The most prominent protocol applying a RTS-CTS handshake is the IEEE 802.11 standard for wireless local area networks. According to [9], the effectiveness of the RTS-CTS handshake is based on the assumption that hidden nodes are within transmission range of receivers. Neither the RTS-CTS handshake nor an extended carrier sensing range are ultimate solutions to solve the hidden node problem, collisions can still occur in both cases, as surrounding noise can additively contribute to the interference, or nodes can simultaneously switch to the transmit state and access the medium. Introducing a RTS-CTS handshake means a high overhead, especially in a sensor networks, where payloads are considerably small. The question whether it pays off to apply a RTS-CTS scheme in a wireless local area network has been intensively studied in [10]. The authors conclude that *considering the fact that under low load condition in the network the few collisions caused by the hidden terminal scenario do not harm the overall performance as much as under high load condition, it seems to be reasonable only to use RTS-CTS when the network load is high.* They argue that RTS-CTS only make sense when the average frame exceeds a certain vulnerable length. As WiseMAC aims for efficiency in low-traffic scenarios, and payloads are considerably small in wireless sensor networks, introducing an extended carrier sensing range is a cheaper and more suitable solution.

2.1.3.2 Nonpersistent-CSMA with preamble sampling

To mitigate collisions caused by concurrent transmissions to the same node or other transmissions in the immediate vicinity, WiseMAC follows a simple CSMA/CA scheme, as done in most unscheduled wireless MAC protocols. With wireless transceivers, collisions are harder to detect than in wired communication, as the power difference between the transmitted signal and the received signal makes it impossible to distinguish the emitted signal from the received signal. Anyway, most wireless transceivers do not permit to send and receive at the same time.
WiseMAC proposes to prepend a medium reservation preamble of limited random length to the actual preamble. Before starting to transmit this preamble, nodes check the carrier and wait for a certain time. When there is more than one station aiming to transmit a packet, the station with the longest medium reservation will start transmitting and win the contention. Collisions and errors in transmission are detected using acknowledgements and are resolved with an Automatic Repeat Request (ARQ) scheme. Collisions can still occur because of the hidden node problem, signal propagation delays and the vulnerable time period when two stations are simultaneously switching their transceivers to the transmit state.
Non-persistent CSMA proposes to reschedule a transmission within a random time value in between uniform [0, MaxBackoff] when the first transmission attempt fails. The combination of non-persistent CSMA and the WiseMAC preamble sampling technique works as follows: a node aiming to transmit a frame schedules the transmission for a certain instant. If at this instant, the node finds the medium busy, it reschedules the transmission to a later instant in between [0, MaxBackoff] and turns its radio to the sleep state for the time in between. When aiming to send a unicast packet to a station that is periodically sampling the medium, the node must choose the next wake-up of the respective destination and try again with another random medium reservation preamble.

2.1.4 Fixed and Random Intervals

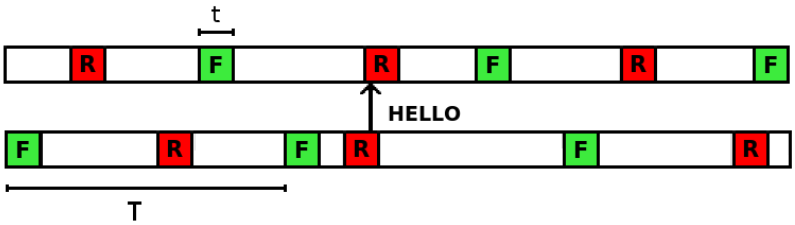


Figure 2.10: Unsynchronized Power Saving Mechanism with Fixed and Random Intervals

The mechanism proposed by *Braun et al.* in [2] and analyzed in a static multihop wireless ad hoc network environment in [3] defines two wake and two sleep intervals during one basic interval duration T. As depicted in Figure 2.10, nodes alternate between the wake and sleep states in their individual wake-up pattern, with their duty cycles remaining fully unsynchronized. The nodes strictly alternate between a fixed wake interval F and a random wake interval R. Each of the wake periods are of same duration t. The fixed wake interval F always starts at the beginning of the slot. The start of the random wake interval R is uniformly distributed between the end of the fixed wake interval F and the start of the next one. Hence, all nodes operate with the same wake ratio $W = 2t/T$.
The fixed wake interval F enables a node aiming to contact any neighboring node, if its periodically occurring fixed wake interval pattern is known. If there is no intersection between the fixed wake interval of the sender and the neighbor, it may never learn about its presence. This motivates the choice for the random secondary wake interval R. It ensures that two nodes with disjoint wake-up pattern will sooner or later be awake at the same time and therefore be able to exchange announcements about their own wake interval. By receiving these, the nodes will learn about the wake-up patterns of their neighbors, and thus be capable to reach any neighboring node during their fixed wake interval F. The mechanisms applies the 802.11-based medium access and contention scheme without the RTS-CTS prefix, as disseminating RTS and likewise CTS control messages is difficult and costly with unsynchronized wake intervals.

In [3], this wake-up scheme is applied to a multi-hop wireless ad hoc network and a reactive routing protocol. In order to efficiently disseminate broadcast messages, [3] suggests to combine and exploit the information about the next soonest wake-ups of each node's neighboring nodes. By figuring out the best instant for sending and forwarding a broadcast, the so-called wireless multicast advantage can be exploited. A node intending to broadcast a message can figure out the best instant to forward the message using previously received wake-up announcements which can be used to determine the current duty cycles. The best instant shall be the instant during the next basic cycle T when the largest subset of the neighboring nodes is awake. The aim is to transmit the message during some neighbor's intersections, if there are. A node can calculate the best instant and duration of possible intersections by exploiting and combining the knowledge about each neighbor's respective wake-up patterns.
Figure 2.11 depicts the concept to search the best instant. The node calculates the best moment for broadcasting a message, and selects any instant in between Δx. If there are more than one possible instants, one intersecting moment shall be chosen at random. The aim of the broadcast is therefore not to reach all of the neighbors, but only the largest possible count of neighbors with each attempt, as it is done in probabilistic broadcasting techniques.

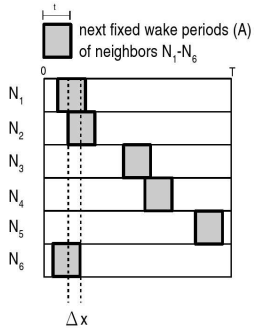


Figure 2.11: Intersection table

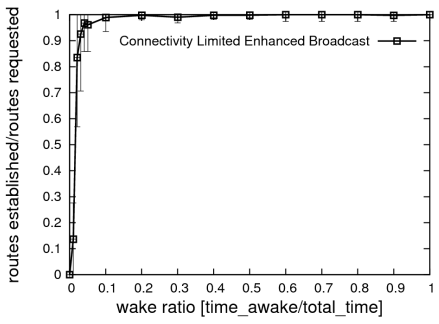


Figure 2.12: Connectivity

This scheme was then tested out to find routes with a AODV Route Request (RREQ)-Route Reply (RREP) query cycle between random senders and destinations on a 1000 m $\times$ 1000 m plane with 200 uniformly distributed nodes. Using this technique, and taking the two best instants for rebroadcasting each RREQ, the success ratio reached 97% even for the very low wake ratio of 4%, as depicted in Figure 2.12. By rebroadcasting a RREQ message maximally twice in every node, the disadvantage of the unsynchronized wake-up pattern in regard of broadcasting becomes negotiable, when considering the efforts that would otherwise be necessary to achieve a rigidly synchronized wake-up pattern.

Furthermore, the unsynchronized wake-pattern led to astonishingly good results in respect to the rate of collisions. As every node waits for another instant to rebroadcast the RREQ, the impact of the broadcast storm problem becomes negligible. Only a very low jitter had to be incorporated for the rebroadcasts of the route request RREQ messages.

The simulation showed that it is possible to sustain a fully unsynchronized wake-sleep schedule on the MAC and reduce the wake-ratio to a few percent. The investigation states that on-demand routing protocols can be applied with a high probability of successful connection attempts on top of this MAC when carefully designing broadcast flood message forwarding strategies.

2.1.5 Scheduled Channel Polling MAC

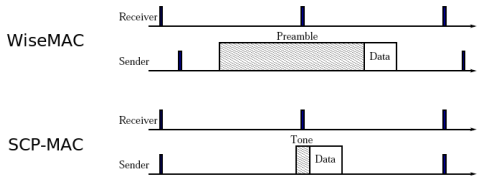


Figure 2.13: WiseMAC vs. SCP-MAC

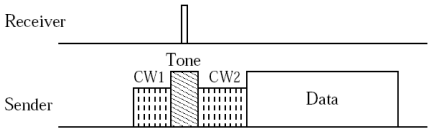


Figure 2.14: two-phase contention

Scheduled Channel Polling MAC (SCP-MAC)[11] is a very recent contribution inspired by the synchronized *scheduled* protocols S-MAC/T-MAC and the unsychronized *unscheduled* preamble sampling based WiseMAC [4].
SCP-MAC combines the advantages of both approaches. Nodes are yet again synchronized as in S-MAC/T-MAC and sample the medium for *channel polls* in a clusterwise or global common wake-up pattern. Nodes that have data to transmit contend in a two-phase contention mechanism. In a first contention window (CW1), they contend for a tone transmission, as depicted in figure 2.14. Nodes winning the contention will transmit a tone. Possible collisions in tone transmission will cause no harm, because only the presence of the tone is important. The receiving nodes sample the medium and operate similarly as in WiseMAC - if there is no tone, the receiving nodes return to the sleep state immediately. If the stations sense the presence of the tone, they remain awake for the next phase. The actual contention for the data transmission then takes place in a second contention window (CW2), but only with the winners of the first contention. In order to avoid collisions from hidden stations, SCP-MAC applies the full RTS-CTS handshake. The sender station announces the transmission and thereby also addresses the receiver node by emitting a RTS. Stations receiving the RTS will know *that* there is a transmission going on and whether they have to stay awake or not. The receiver will likewise emit a CTS and reserve the channel.

There are advantages and drawbacks when comparing the scheme with WiseMAC. Since SCP-MAC synchronizes all nodes to a common wake-up pattern, there is no need for additional bookkeeping of the relative schedule offsets of all neighbors as in WiseMAC. SCP-MAC supports broadcast very well because of the common wake-up pattern, while WiseMAC must use long preambles to reach all neighbors with one message. On the other hand, WiseMAC gets along without a big contention window, because traffic does not only take place in a single common active period, but is spread out over the complete sleep-wake cycle.
The synchronization overhead for keeping the tight sampling scheme for all nodes in SCP-MAC is another concern. The authors propose to piggyback synchronization information to every data packet that has to be sent, and claim that *with data rate higher than synchronization period, piggybacking can completely suppress explicit SYNC messages.*
Another drawback is the inherent latency of packets routed over multiple hops, as each node will have to wait for the full next interval to forward the packet. Depending on the chosen interval rate and the network size, it can take much time for a packet to be routed to a sink. Yet another concern is the initial deployment of the common wake-up pattern. The problem that multiple synchronization clusters evolve arises, as observed in the S-MAC protocol (see 2.1.1). Bordering nodes between two synchronization will have to adopt both wake-up patterns, which is however not that harmful in SCP-MAC, as the duty cycles are very short and not too costly.

2.2 Routing in Wireless Sensor Networks

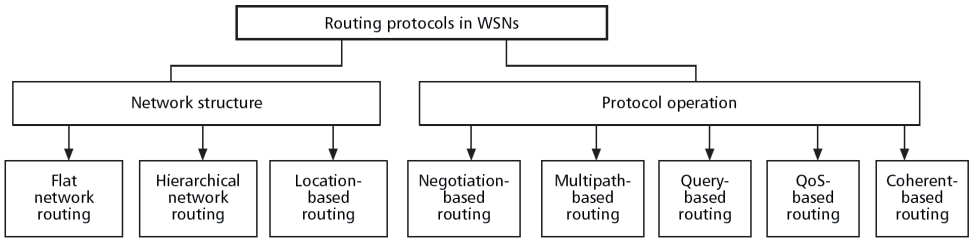


Figure 2.15: Routing protocols taxonomy

Wireless sensor network nodes are severely constrained in energy reserves and bandwidth. Typically, nodes are deployed in a large number. Coordination and management of the sensing activity represent a huge challenge to sensor networks mechanisms designers. Seamless collaboration among sensors is necessary for gathering and reliably processing sensor data to the sink. These challenges necessitate energy-awareness at all layers of the underlying networking protocol stack. At the network layer, methods for energy-efficient route setup and reliable relaying of data from sensor nodes to sink have to be designed. A lot of development in the research on routing protocols for ad-hoc and sensor networks has been seen since the introduction of Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV), Ad Hoc On-Demand Distance-Vector Routing (AODV) [14] and Dynamic Source Routing (DSR) [15]. Countless new routing protocols for sensor networks have evolved since then, laying their focus on new paradigms and introducing various ideas and features. AODV and DSR are nowadays very popular and widely accepted ad hoc routing protocols, several variations of DSR and AODV have been suggested.

Karaki and *Kamal* suggest in [16] to divide the routing schemes for wireless sensor networks according to the network structure into flat-based routing, hierarchical-based routing, and location-based routing, as illustrated in figure 2.15. In flat-based routing, all nodes are typically assigned equal roles. In hierarchical-based routing, nodes have different roles. Often, nodes are organized into clusters so that so-called cluster heads aggregate and preprocess gathered data and thereby save some energy. Location-based protocols utilize the position information to relay the data to the desired regions rather than the whole network. The sensor nodes' positions are exploited to route data in the network. Depending on the protocol operation, [16] classifies sensor networks protocols into multipath-based, query-based, and negotiation-based, QoS-based, or coherent-based routing techniques.
In regard of how and when routes are discovered and computed, routing protocols are classified into the three categories, proactive, reactive, and hybrid, depending on how the source finds a route to the destination. In proactive protocols, all routes are computed before they are really needed, while in reactive protocols, routes are computed on demand. Hybrid protocols use a combination of these two ideas. A significant amount of energy is used in the route discovery and path setup phase of routing protocols.

In the following, we discuss the two protocols AODV and DSR. The protocols both follow the flat rather than the hierarchical routing paradigm. Both protocols are reactive and calculate paths only when they are actually needed. The two protocols seemed suitable to operate with the asynchronous MAC mechanisms examined and developed in this thesis, which neglect to build clusters and assign special functions to single instances on the MAC. We decided to pursue the flat-based paradigm on the routing layer too.

2.2.1 Ad Hoc On-Demand Distance Vector Routing

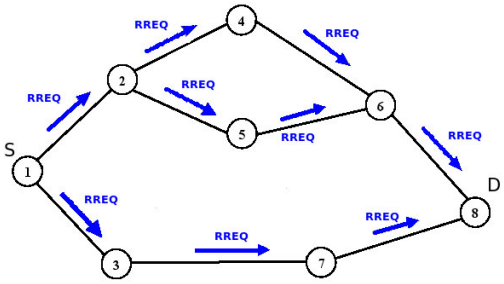


Figure 2.16: AODV Route Request

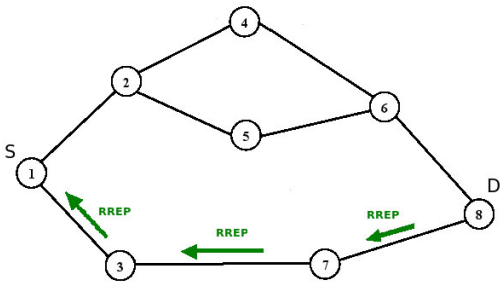


Figure 2.17: AODV Route Reply

The Ad Hoc On-Demand Distance Vector Protocol (AODV) [14] is an on-demand routing protocol for ad hoc networks and mobile ad hoc networks. It builds routes using a route request (RREQ) and route reply (RREP) query-response cycle. Each RREQ contains a source sequence number and a destination sequence number. The source sequence number is an indicator of the route information freshness. Every node maintains an own sequence number and increases it with every RREQ broadcast it originates. The destination sequence number denotes the last received sequence number of a destination node.

When a source node requires a route to a certain destination, it floods a RREQ packet across the network, given a contiguous network topology. Any nodes receiving the RREQ update their table entry for the source node and set up entries to the neighboring sender node, if they provide the freshest and shortest path to the source node. Duplicate incoming RREQ's are identified using the sequence numbers and are simply discarded.
If an intermediate node receives the RREQ, it sends a RREP if it has a *fresh enough* routing entry to the destination of the RREQ in its table. It responds in place of the destination node by unicasting a RREP towards the source to shorten the route discovery process and limit the flooding. The RREQ is then not further forwarded. If the intermediate node has no valid entry, it simply rebroadcasts and forwards the RREQ.
When the RREQ passes through and reaches the destination, the destination responds with a RREP. If a later incoming duplicate of the RREQ proposes a shorter route to the sender node, the destination node responds again by sending another RREP to the corresponding node.
To keep their routing tables up to date with information about the current neighboring nodes, AODV proposes to broadcast HELLO messages. HELLO messages are intended to maintain local connectivity and are not forwarded, or optionally forwarded up to a certain defined distance.

When a node detects a broken link while attempting to forward a packet to the next hop, it generates a RERR packet that is either sent to the node previously having sent the packet and is forwarded to the originating node, or to all neighboring nodes. The RERR packet erases all route entries for this destination and serves as notification for the link break. If a source receives a RERR packet and a route to the destination is still required, it initiates a new route discovery cycle. Routes can also be deleted from the routing table if they are unused for a certain amount of time (route aging).

2.2.2 Dynamic Source Routing

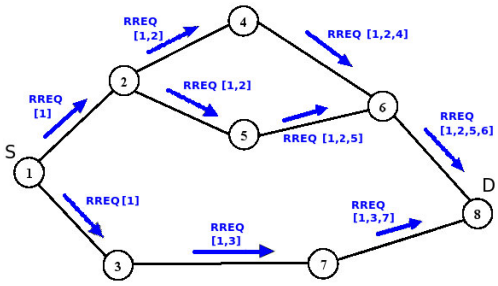


Figure 2.18: DSR Route Request

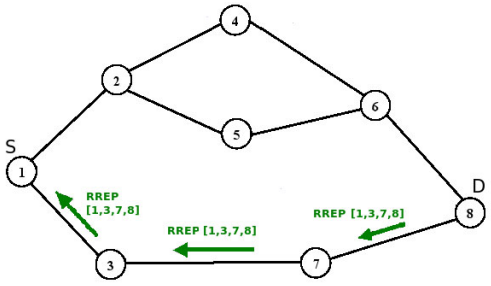


Figure 2.19: DSR Route Reply

Dynamic Source Routing (DSR) [15] is an on-demand routing protocol for ad hoc networks and mobile ad hoc networks. Like any source routing protocol, DSR includes the full route in the packet header. Instead of relaying on a routing table, each intermediate node only needs to forward the packet to the next node on the route in the packet. Intermediate nodes do not necessarily have to maintain local routing information, what may be considered as an advantage compared to table-driven protocols.

When a node aims to send a message to a destination it has no route for in its route cache, it broadcasts a route request (RREQ) message towards the destination, which is in turn flooded across the network. The RREQ message includes the route specifying the sequence of nodes the message has already traversed, as illustrated in figure 2.18, and a sequence number generated by the source node. After reception of a RREQ, an intermediate node checks to see if it is already contained in the route record. The sequence number on the packet is compared to the respective numbers of recent RREQ's in the cache in order to recognize duplicates. If the receiver is already in the route record, the intermediate node drops the message, in order to prevent routing loops. If not, and if the packet is not a duplicate, it is forwarded and rebroadcast.
Intermediate nodes passively collect the routing information of the RREQ header in their route cache for their own possible future use. When the destination receives the RREQ from a source it has a route for in its route cache, it sends a route response (RREP) message along this route.
Intermediate nodes may optionally use their route cache to reply to RREQ's. If an intermediate node has a route to the destination in its cache, it appends the route to the route record in the RREQ and sends a RREP back to the source containing this route, in order to limit the flooding of the RREQ. However, if the cached route is outdated, this can result in the source receiving inactive routes.

When a node detects a broken link while trying to forward a packet to the next hop, it sends a route error (RERR) message back to the source along the same route. When an RERR message is received by intermediate nodes and the source node, all respective routes containing the broken link are deleted.

2.3 Multipath Routing

Standard routing protocols in ad hoc wireless networks, such as AODV [14] and DSR [15], are mainly intended to discover one single route from a source to a destination. During the route discovery process, these protocols aim to find the best route that advertises the lowest cost.
Multipath routing protocols aim to find multiple routes. Multiple routes can be useful to compensate for the dynamic and unpredictable nature of ad hoc networks, supposedly also in energy and bandwidth constrained sensor networks. Multipath Routing is discussed in the context of both wired and wireless communication networks. It has been proposed for application in the internet, in metropolitan and local area networks, in wireless mobile ad hoc networks, as well as in wireless sensor networks.
Mueller and *Ghosal* discuss in [18] common goals, problems and recent suggestions for multipath routing protocols in wireless ad hoc networks. We focus in this section only on reactive multipath routing schemes. We begin by outlining the main benefits of multipath routing, discuss problems that may arise when applying such schemes to wireless ad hoc networks and sensor networks and conclude with a brief overview of recent work on multipath routing protocols in wireless channel networks, laying our focus on the application in wireless sensor networks.

2.3.1 Benefits of Multipath Routing

Discovering and maintaining multiple paths requires a certain overhead, but yields conceptual advantages. The following goals are typically addressed when applying multipath routing schemes:

- *Load Balancing:* Multipath routing provides a means to avoid and resolve congestion and to improve QoS characteristics. When certain nodes and links become over-utilized and cause congestion, multipath routing can be applied to spread the traffic over alternate paths, thereby balancing the load over both paths. In wireless sensor networks, the main focus of multipath routing is typically on the load balancing aspect. As nodes are constraint to a limited amount of energy, and traffic is expected to be low, the main concern is to keep the network operable for a maximum amount of time. In sensor networks, one has to deal with traffic generated by many leaf nodes attempting to deliver data to one or a few sinks. Usual on-demand routing schemes tend to utilize always the same set of nodes to forward packets, whereas many other nodes stay unused. It has been observed that in such a case, nodes that have to forward traffic from large subtrees drain much earlier due to energy depletion, whereas other nodes have only slightly been used. When nodes collaborate in the sensing and data forwarding activity more equitably, and packets are not always routed on the same routes, but balance the load over multiple routes, network lifetime can be substantially increased.

- *Fault Tolerance:* Multipath routing protocols provide measures to increase the degree of fault tolerance by having redundant information routed to the destination over alternate paths. This increases the energy overhead, but helps to reduce the probability that communication is disrupted and data is lost in case of link failures. Sophisticated algorithms have been developed to increase the degree of reliability. The trade off between the additional overhead and the reliability gain has been subject to recent research [17]. As fault tolerance and reliability is not the main topic of this thesis, we omitted further investigations on this issue.

- *Bandwidth Aggregation:* By splitting data to the same destination into multiple streams, each routed through a different path, the effective bandwidth can be aggregated. This strategy is especially beneficial when a node has multiple low bandwidth links but requires a bandwidth that is greater than each individual link can provide.

- *Reduced delay:* For wireless networks employing single path on-demand routing protocols, a route failure means that a new path discovery process needs to be initiated to find a new route. This results in a route discovery delay. The delay can be minimized in multipath routing, as backup routes can be identified and held active during one single route discovery attempt. Furthermore, discovering several paths and observing QoS characteristics of both paths permits to switch the load to another route whenever the service parameters of another route promise better throughput and lower latency.

In wireless sensor networks, the focus of multipath routing often lies on the load-balancing or the fault-tolerance aspect, rather than on the aggregation of bandwidth. Often, the goal of multipath routing protocols is to maximize the time the network is operable and fulfills its observation task. A brief overview over concepts of network *lifetime* is given in section 2.4.

2.3.2 Route Coupling

Using multiple paths in ad hoc networks to achieve higher bandwidth, balance load or achieve fault-tolerance is not as easy as in wired networks. As nodes in the network communicate through the wireless medium, radio interference must be taken into account. Transmissions from a node along one path may interfere with transmissions from a node along another path, even if the paths are link- or even node-disjoint. The interference may limit the achievable throughput and lead to two paths hindering each other in forwarding packets. This phenomenon is often referred to as *route coupling*. Route coupling occurs when two routes are located physically close enough to interfere with each other during transmission. As a result, the nodes in those two routes are constantly contending for access to the same medium. The advantages of two routes being available are therefore narrowed.
Pearlman et al. discuss in [19] that in wireless networks, route coupling caused by radio interference between paths can have serious impacts on the performance of multipath routing protocols, even if the paths are disjoint. In some cases, route coupling can even lead to worse results than routing over one single path. The shared transmission medium forces all nodes in the interference range of a sender to remain silent until completion of a transmission, and the problem even gets worse when applying a RTS-CTS handshake.

Waharte and *Boutaba* [21] study the influence of route coupling in wireless channel networks applying multipath routing, and distinguish between routes that have a) no common collision domain, b) routes with a link connection in between and c) nodes sharing a common node.

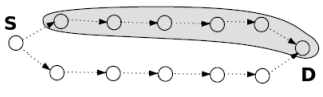


Figure 2.20: a) without connection

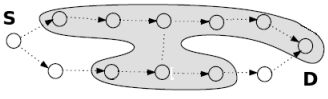


Figure 2.21: b) link connection

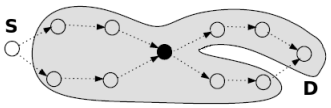


Figure 2.22: c) node connection

It is important to notice that node-disjoint paths, which are preferable in multipath protocols, do not necessarily have to be of type a). Even if the nodes of the paths form disjoint sets, each of the intermediate node might still be linked to nodes of another path, thereby causing the problem of route coupling by tampering transmissions going on on other routes. The authors conclude that paths of type a) produce the best throughput results, as the common collision domain of the multiple paths is minimized to source and destination nodes, and transmission along the path are independent to the largest possible extend. [21] concludes that *although a more efficient network utilization due to a better load balancing can justify the use of a multipath routing strategy compared to single path routing, the benefits of multipath routing in terms of throughput quickly vanish when interference is accounted for.*
The authors of [22] tend towards a similar conclusion. They argue that the common multipath protocols mainly find routes that are too close to each other to actually behave much different than single path routing schemes, and conclude that to actually make energy savings, multipath routes would have to make sure that traffic is routed along routes that do not interfer with each other at all, which is - in most cases - illusional. Factually, none of nowadays established and well-investigated proposals have considered and incorporated the route-coupling phenomenon for effective load balancing.
Recent research has been pursued on the issue of on-demand construction on non-interfering multiple paths in sensor networks in [20]. The proposed mechanism routes packets along paths that have a gap of two transmission ranges in between. The mechanism however strongly relies on the position-awareness of the sensor nodes and the knowledge of the position of the receiver.

2.3.3 Example Multipath Routing Protocols

2.3.3.1 Split Multipath Routing

Split Multipath Routing (SMR) [25] is an on-demand multipath source routing protocol. As done in every source routing protocol, SMR includes the full route in the route discovery control packet header. Intermediate nodes use the received routing information to forward packets towards the destination, and to furthermore extract knowledge about the network topology and build up a route cache containing routes to other nodes.
In general, SMR is quite similar to DSR, and aims to obtain maximally disjoint paths by a classical Route Request - Route Reply query cycle. Unlike in DSR or AODV, the intermediate nodes do not reply to incoming RREQ's to shorten the RREQ query cycle, as the destination shall receive all the routes, and select the maximally disjoint paths that have the fewest links or nodes in common.
Duplicate RREQ's are not always discarded. Instead, intermediate nodes forward all incoming RREQ's that are received through a different incoming gateway node. They also forward RREQ's when the hop count is not larger than the respective hop count of previously received RREQ's.
The SMR route selection algorithm is designed to discover two routes, but can be extended to find more than two routes, if there physically are. The destination sends a RREP for the first RREQ it receives, which is expected to be the path with the shortest delay. The destination node then waits to receive more RREQ's. From the later received RREQ's, the path that is maximally disjoint from the first discovered path is selected. If more than one maximally disjoint path exists, the shortest path is selected. If more than one shortest path exist, the path whose RREQ was received first is selected. Finally, the destination sends a RREP for the RREQ providing the best alternate path towards the source, thereby establishing the full bidirectional route.

2.3.3.2 Ad Hoc On-Demand Multipath Distance Vector Routing

The Ad hoc On-Demand Multipath Distance Vector Protocol (AOMDV) [26] is an extension to the AODV protocol for discovering node-disjoint or optionally link-disjoint paths. It achieves to finds node-disjoint paths by exploiting a particular property of flooding. By appending the *first-hop* to the RREQ header, and bookkeeping about the first-hops of the recently received RREQ's, nodes receiving duplicate RREQ's by different neighboring nodes can easily determine if the routes are node-disjoint. The first-hop is the first node a RREQ traverses after the initiating source. To find node-disjoint routes, nodes do not immediately reject RREQ's. Each RREQ arriving via a different neighbor of the source has another first-hop in the RREQ header, and therefore defines a node-disjoint path. This is because nodes do never rebroadcast duplicate RREQ's, so any two RREQ's arriving at an intermediate node via a different neighbor of the source could not have traversed the same node. As in AODV, RREQ duplicates are discarded in intermediate nodes. RREQ's with equal destination sequence number, but incoming from another gateway node are simply ignored in AODV, unless they provide a better hopcount value. In AOMDV, intermediate nodes, as well as destination nodes do reply to such RREQ's if their first-hop is different from the one prior received, and reply to them with RREP messages. Using this policy, AOMDV guarantees node-disjoint paths whenever it takes up a second routing entry to the same destination. AOMDV further allows to discover link-disjoint path by exploiting RREQ duplicates coming in at the destination via different gateways. AOMDV [26] leaves the choice if the option shall be used open *to the implementer or even the user.*

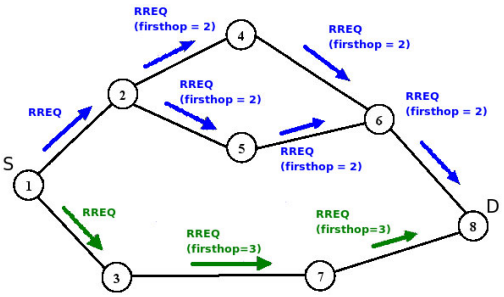


Figure 2.23: AOMDV Route Request

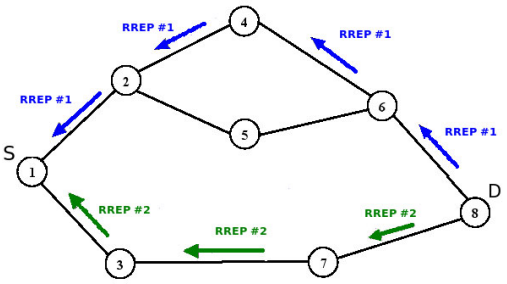


Figure 2.24: AOMDV Route Reply

routing table node #1			
dest	next	hops	seq
8	3	3	37
8	2	4	37

routing table node #6			
dest	next	hops	seq
1	4	3	11
8	8	1	37

routing table node #8			
dest	next	hops	seq
1	7	3	11
1	6	4	11

Figures 2.23 and 2.24 illustrate the AOMDV mechanisms to find node-disjoint paths. The illustration shows node 1 initiating a route request to node 8. The RREQ is flooded over node 2 and node 3. There, the *first-hop* field is set accordingly. The RREQ's finally reach the destination node 8, where both incoming requests provoke new path entries for the source node 1, as the incoming RREQ packets exhibit a different first-hop. Furthermore, to establish the full bidirectional routes, both RREQ packets are replied. Node 6 similarly receives two path RREQ packets over the two nodes 4 and 5. Both RREQ's however exhibit the same first-hop. Node 6 therefore knows, that the paths to the source node advertised by these RREQ packets are not node-disjoint, and does not add a second path entry.
To support multipath routing, the AOMDV route tables contain a list of gateways and

hop counts for each destination node. The path entries to a destination have all the same destination sequence number, as they have been obtained in one single RREQ-RREQ query cycle. When receiving a path advertisement with a higher sequence number, all routes with the old sequence number are removed.

2.3.3.3 Ad Hoc On-Demand Distance Vector Multipath Routing

The Ad Hoc On Demand Distance Vector Multipath Routing Protocol (AODVM) [27] is another extension to AODV for finding multiple node-disjoint paths. In AODVM, intermediate nodes are not allowed to shorten the RREQ and send a route reply to the source, and duplicate RREQ packets are not discarded by intermediate nodes. For each received copy of an RREQ message, the receiving intermediate node records the source who generated the RREQ, the destination for which the RREQ is intended, and the neighbor who transmitted the RREQ in the so-called RREQ table.
When the destination receives the first RREQ packet from one of its neighbors, it generates a RREP packet. The RREP packet contains an additional field *last hop* to indicate the neighbor from which the particular copy of RREQ packet was received. This RREP packet is sent back to the source via the path traversed by the RREQ copy, to establish the full bidirectional route. When receiving duplicate copies of the RREQ packet from other neighboring nodes, it sends RREP packets to each of these neighbors. Like the first RREP packet, these RREP packets also contain the *last hop* that the corresponding RREQ took. When an intermediate node receives an RREP packet from one of its neighbors, it deletes the entry corresponding to this neighbor from its RREQ table and adds a routing entry to its routing table to indicate the discovered route to the originator of the RREP packet. By checking the RREQ table, the intermediate node figures out the neighbor with the shortest path to the source and forwards the RREP message to that neighbor. The entry corresponding to this neighbor is then deleted from the RREQ table. As long as there is an entry in the RREQ-table, incoming RREP's are forwarded towards the source.

To ensure that nodes do not participate in multiple paths, nodes have to monitor the outgoing traffic of their neighboring nodes. When a nodes overhears a RREP packet emitted by a neighboring node, the corresponding entry is deleted from the RREQ tables. Because a node cannot participate in more than one route, the discovered routes must then be node-disjoint.
In AODVM, the intermediate nodes make decisions where to forward the RREP messages. The destination node originating the RREP's does not know if and how many of these RREP messages actually get back to the source. Thus, it is necessary for the source to confirm each received RREP message by means of a Route Confirmation message (RRCM). The RRCM message is piggybacked on the first data packet sent on the corresponding route, and optionally contains further information such as the hop count of the route, or the first and last hop nodes.

2.3.3.4 Discovering and Maintaining Braided Paths

Ganesan et al. [24] consider the question how to construct the secondary paths which are - in the optimal case - node disjoint, but focus their study on the question how to keep the overhead as small as possible if only one node or one link in the network fails. The authors argue that when a small number of multipaths are kept alive, failures on the primary path can usually be recovered without invoking network-wide flooding for path discovery. This feature is important in sensor networks since flooding is very costly and can vastly reduce network lifetimes.
Node-disjointness is a very strong condition when aiming to find multiple paths between

two nodes. It seems to produce rather inefficient and suboptimal paths in terms of hopcount. Long detours around many nodes can be necessary to fulfill the condition of node-disjointness. Alternate node-disjoint paths can become very long, and therefore expend significantly more energy than expended on the primary path.
To overcome this problem, and yet retain the robustness advantages of multiple paths, the authors suggest the construction of so-called *braided* paths. Braided multipaths relax the requirement for node-disjointness. Such paths are only required to leave out some of the primary path's nodes. They are free to use other nodes on the primary path. The authors propose to construct two different kinds of redundant paths - node-disjoint paths and *braided* paths. Which of the two schemes to use shall depend on the failure patterns. They measure the so-called path *resilience* as the percentage of paths that is compensated by a so-called alternate path reinforcement to reconstruct a failing path. The authors claim a overall better path resilience with the braided path approach.

2.3.3.5 Probabilistic Routing over Suboptimal Paths

When discovering and maintaining multiple paths from a source to a destination, it may make sense also to occasionally use suboptimal paths in terms of hop-count that use more energy for an end-to-end transmission than the optimal one. Traffic load can be spread over multiple paths, which leads to more nodes participating in the forwarding process. Using the lowest energy path for all packets is not necessarily best for the long-term health of a sensor network, as important forwarders might drain out of energy first.

Shah and *Rabaey* [28] suggest a quite simple approach to probabilistically incorporate suboptimal routes. Each node maintains an energy cost estimate for each of its path entries. This cost estimate determines the probability that a packet is routed over this certain path. If a node aims to transmit a packet to a certain destination it has multiple paths for, it chooses the forwarding node according to a probability assigned to that path. Each of the intermediate nodes do the same and forwards packets according to the probability assigned to the different paths in their table, if there are more than one. This is continued until the data packet reaches the destination node. The probability assigned to a gateway N_i to destination N_j is weighted to be inverse proportional to the cost over this gateway in relation to the sum of the costs of all known paths to this destination. When aiming to transmit a packet to N_j, a node chooses the path over N_i with the following probability:

$$P_{N_j,N_i} = \frac{\frac{1}{C_{N_j,N_i}}}{\sum_{k \in FT_j} \frac{1}{C_{N_j,N_k}}}$$

Where FT_j is the set of all path entries destined to N_j. Consider the routing table of node 1 in section 2.3.3.3. Node 1 knows two paths towards node 8, with either nodes 3 or 2 as gateways with costs 3 or 4, respectively. When aiming to transmit a packet to node 8, the probability with which the packet is routed over 3 is $P_{3,8} = \frac{\frac{1}{4}}{\frac{1}{4}+\frac{1}{3}} = 0.57$ Using this simple mechanism to send traffic over different routes helps in using the nodes' resources more equitably. The authors test the probabilistically weighted forwarding strategy in a sensor network scenario, and measure the lifetime as the time to the first node depletion. They claim an overall gain of $\sim 40\%$ of network lifetime increase with this probabilistic routing scheme. The authors argue that indeed, taking suboptimal paths occasionally into account is strategy that pays off as nodes use their scarce resources more equitably, which helps to disburden central forwarder nodes that would otherwise drain out of energy first.

2.4 Network Lifetime Metrics

There is no single answer to the question how we can measure the lifetime of a sensor network. One has to make more assumptions about when the network can be considered as operable and fulfilling its purpose and when not, or not anymore. There is no consistent definition for the *lifetime* of a sensor network within the literature. Many researchers have defined their own unique measures of network lifetime in their investigations on different topics. Network lifetime must be defined for each application purpose of a sensor network. In some cases, coverage is a very critical issue, and the depletion of one single node may already make the network much lesser useful. In other cases, the loss of half of the nodes might still be bearable. Node density and available redundancy usually makes a big difference.
Karl and *Willig* pointed out the most common approaches in [23]. A brief overview about the network lifetime metrics pursued in recent wireless network sensor lifetime evaluations is given below.

- *First Node Depletion*: One simply measures the time between deployment and initiation of the sensor network and the first depletion of a sensor. This approach is easiest to implement. A big part of the publications on wireless sensor network MAC and routing schemes that target at lifetime-optimization defines network lifetime in this manner, especially when coverage is a big concern, and not much redundancy is supplied. In many other applications, this definition makes fewer sense. When dealing with wireless sensor networks with countless redundantly deployed nodes, the loss of one node can be considered irrelevant.

- *Network Half-Life*: Fewer publications defined the network lifetime as the time until 50% of the nodes drain out of energy. Applying this metric opens the problem how to continue when a network gets partitioned into disjoint sets without any interconnection. In some cases, one would expect such a network not to be operable any more. But if one considers that recollecting the nodes is possible, and no in-time-delivery of data is necessary, this metric makes sense. In other cases, researchers proposed to measure the time to the depletion of the first 10%, 20% or else percentile of nodes as a metric for the network lifetime.

- *Network Partition*: The time to the first network partition has also been suggested as a network lifetime metric. One measures the time until the network is split into two or more disjoint network partitions due to the depletion of pivotal interconnecting nodes. This measure is very dependent from the network toplogy. If there are a few bottleneck nodes that connect two or more clusters of nodes, the measure might indicate the network as *dead* very early, whereas in robust topologies with many redundant nodes, it might take much longer.

- *Hybrid Metric*: Hybrid definitions of the upper conditions can also be found in the literature on sensor networks. Some researchers defined the lifetime as the time until some percentile of the nodes depletes *or* the network becomes partitioned. When there is only one sink, a network partition immediately leads to one partition of nodes that is no longer in reach of the sink and can thus not forward data anymore.

Obviously, defining the network lifetime is a critical task. If the performance of different sensor network mechanisms is measured, simulations should be carried out with different network lifetime metrics, in order to omit that the obtained results are not a consequence of the scenario setup and the chosen the lifetime metrics. Applying different metrics underlines and proves the robustness and significance of the obtained results.

2.5 Convergecast and Data Aggregation

Data aggregation techniques have been heavily examined in the past few years. Data aggregation and in-network-processing are nowadays established paradigms for wireless sensor networks. Data from remote stations can be preprocessed or aggregated in intermediate nodes, in order to eliminate redundancy, minimize the number of transmissions and thus save energy. Incoming messages from different stations can be converged to save some MAC and routing headers transmission overhead. Several suggestions propose to preprocess data on the application level, such that intermediate nodes filter, extract and compact application layer data to minimize the transmissions. There have been suggestions for intelligent query dissemination, where queries are flooded and disseminated in the network, and responses are aggregated and preprocessed by intermediate nodes. Finally only the results are transmitted back to the sink.
There are many different approaches to tackle the data aggregation and preprocessing problem. Data aggregation can generally be considered as an optimization problem. In [32], the authors define a model with several constraints, an initial energy endowment and costs for every transmit/receive operation. The authors then calculate an optimal solution by means of a linear programming approach, which however relies on the assumption of global knowledge about every node's position and energy supplies.
In the following, we briefly discuss two approaches of data aggregation techniques. The first approach proposes to first get a complete picture about the network topology at one central node, then propagate schedule assignments to every leaf node in case of broadcast-initiated convergecast traffic. The second approach is of more distributed nature, and consists in nodes independently aggregating data packets and buffer them for a while before sending it further.

2.5.1 Tree-based Convergecasting

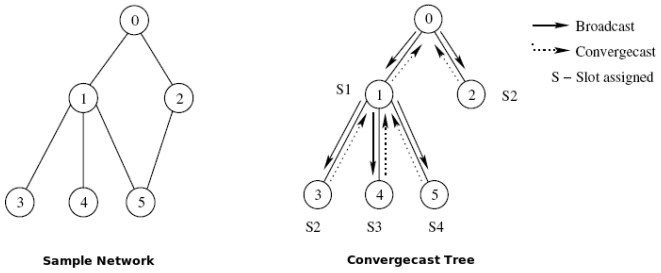


Figure 2.25: Tree-based Convergecasting

Several protocols schedule transmissions along a spanning tree in order to avoid collisions and interferences. As shown in [30], the problem of minimizing the total transmissions along a minimum spanning tree is assumed to be NP-hard. *Gupta* et al. proposed a mechanism in [31] to construct a spanning tree. The mechanism assigns schedules for each link, aiming to minimize the total delay in a broadcast-convergecast query. The heuristic *Convergecasting Tree Construction and Channel Allocation* algorithm is run by the root node and assigns to each node a schedule in a breadth first tree traversion for collision free convergecasting. The mechanism focuses and relies on the communication pattern of a initially injected broadcast query and a convergecast of the responses of all leaf nodes of the network. The mechanism is not intended for event-based traffic, where nodes independently initiate data transmissions.

2.5.2 Data Combining Entities

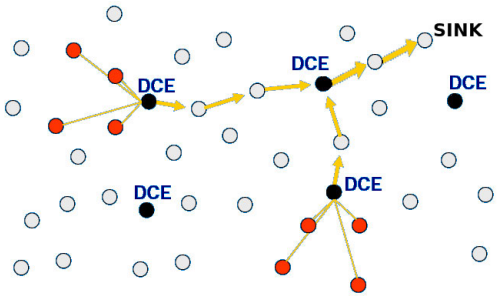


Figure 2.26: DCE nodes combining and aggregating data

Schurgers et al. [29] discuss the introduction of so-called Data Combining Entities (DCE). DCE nodes are designated nodes behaving different than the other neighboring nodes. They buffer incoming messages and wait for a certain predefined time interval before forwarding them. They wait for other incoming packets with the same destination. When more packets arrive, DCE nodes combine the payload or even preprocess the payload on the application layer, and forward them together as one packet to save the header overhead. This approach introduces new delays when buffering the packets. It may depend on the application type whether the increase of delay is tolerable in order to save some energy and maybe prolong the lifetime or not.
The choice of appropriate DCE's in a network may by achieved by sophisticated algorithms to figure out spots with high incoming traffic and high fan-in. As pointed out by *Karl* and *Willig* in [23], aggregation should happen close to the sources, and many sinks should be aggregated as early as possible. As the problem to find these nodes along a minimum spanning tree is also known to be NP-hard and implies total topology information in one node, searching for optimal aggregation points might become a costly issue.

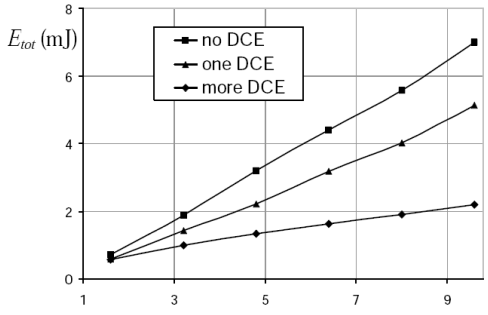


Figure 2.27: Energy consumption with increasing traffic

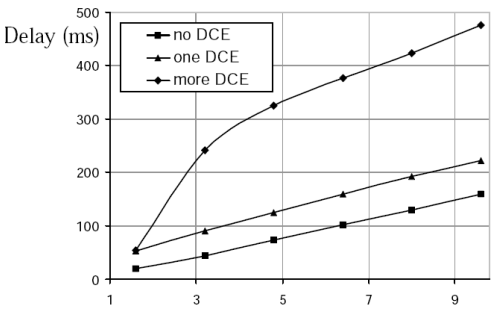


Figure 2.28: Delay with increasing traffic

Figures 2.27 and 2.28 depict the results of the examinations in [29] when applying node-to-sink traffic of increasing rate to a wireless sensor network scenario with *no*, *one* or *more* DCE nodes. As expected, the mechanism allows to improve the energy efficiency. The total energy consumption decreases for the same traffic with more DCE nodes deployed, at the cost of an increasing end-to-end delay introduced by the buffering.

Chapter 3

Experimentation Testbeds

3.1 Network Simulator Environment

We simulated different wireless sensor network scenarios with the OMNeT Network Simulator [34] and the Mobility Framework from TU Berlin [35]. The Mobility Framework is a framework to support simulations of wireless ad hoc and mobile networks on top of OMNeT. This framework incorporates a sophisticated transmission model which is based on calculation of SNR (Signal-to-Noise Ratio) and SNIR (Signal-to-Noise-and-Interference Ratio) values according to a restricted free space propagation model.

The energy consumption model is based on the amount of energy that is used by the transceiver unit. With the parameters below, we do not take processing costs of the CPU into account. Each node's energy consumption is calculated in respect to the time and input current that the nodes spend in the respective operation modes idle/recv, transmit and sleep. Furthermore, state transition delays are incorporated to model the state transition lag and the respective costs.
In a later investigation on the sensor testbeds, we determined the relation between the energy consumption in the different transceiver states which deliver a more realistic energy model of the ESB nodes. The parameters are listed in section 3.3. If not explicitly mentioned, the simulations base on the parameters below.

simulation parameters	
path loss coefficient α	3.5
carrier frequency	868 MHz
transmitter power	0.1 mW
SNR threshold	4 dB
sensitivity	-101.2 dBm
sensitivity carrier sensing	-112 dBm
communication range	50 m
carrier sensing range	100 m

3.1.1 Signal Propagation Model

The Mobility Framework [35] bases on a so-called Free Space Propagation Model, which calculates the received power P_r on a node at distance d to

$$Pr(d) = \frac{Pt\lambda^2}{(4\pi)^2 d^\alpha}$$

where P_t is the transmitted signal power, λ the wavelength of the signal and α the path loss coefficient. If the signal-to-noise (SNR) ratio of the received signal is above a certain threshold, it is assumed that the station can successfully recognize the emitted signal.

As signal dispersion is circular and equal in every direction, all receivers are within a circular transmission range. If there is no other transmission going on in their vicinity and the signal-to-noise ratio is above the threshold during the entire packet transmission, stations within this range correctly receive all frames. The stations out of this range also receive the signal according to the upper equation, but the frames are assumed to be unrecoverable. The received signal then only contributes to the noise level.
The radio propagation model does not take multipath propagation or doppler effects into account, but allows to adjust the path loss coefficient α. Recent examinations of the signal attenuation in IEEE 802.11-based networks [36] conclude that a path loss coefficient between 3 and 4 is most suitable to model wireless propagation in office buildings and outdoor areas. We based our simulations on a path loss coefficient of $\alpha = 3.5$, as many other sensor network simulations base their model on the same or similar values.

3.1.2 Energy Model / Transceiver Model

For performance-evaluation of power-saving MAC and routing approaches, one has to carefully model the transceiver's energy consumption in its respective operation modes and state transition phases, as well as the transition delays and their respective costs.
Feeney and *Nilsson* model the energy consumption of a IEEE 802.11 wireless device in [38] in respect to the transceiver states sleep, idle, receive and transmit. They assume that the per-packet energy consumption can be modelled as

$$Energy = m \times size + b$$

Their model assumes a fixed component b associated with transceiver state changes and channel acquisition overhead (contention mechanism) and an incremental component which is proportional to the size of the packet. The experimental results in [38] confirm a quite high accuracy of the linear model.
The investigations conclude with average values for the power consumption in the different transceiver states. The authors measure 10 mA for the sleep state, 156 mA for the idle state, 190 mA for the receive state and 284 mA for the transmit state. The difference between idle and receive state results from the signal processing on the transceiver circuitry. If the received signal strength indication (RSSI) falls beneath a certain threshold, IEEE 802.11 wireless devices consider the medium as *idle* and do not attempt to reconstruct the signal.

Many low-power and low-bandwidth transceivers used in sensor networks are of very low complexity. Many of them do not even incorporate a digital RSSI indicator. In most cases, the energy consumption in idle and receive mode is approximately equal. The TR1001 transceiver [44] as an example uses the same amount of energy when in the receive state, whether it is receiving a frame or not. It constantly tries to reconstruct a signal, even if there is only Gaussian electromagnetic noise in the channel. When only examining the transceiver's energy cost, and not the processing cost of the microprocessing unit or on-chip signal-processing, one can neglect to distinguish between idle and receive states, as done also in several other investigations on wireless sensor MAC protocols, such as WiseMAC [5] or T-MAC [7]. We sticked to an energy consumption and state transition model with three operation modes sleep, receive and transmit, and applied the respective energy consumption values and state transition delays of the manufacturer [44]. The following table highlights the input current and the state transition delays of the simulations in the next chapter.

supply voltage:	3 V
current:	
send	12 mA
recv	4.5 mA
sleep	5 μA
state transition delays:	
recv to send	12 μs
send to recv	12 μs
sleep to recv	518 μs
recv to sleep	10 μs
send to sleep	10 μs

The energy consumption during the state transition is assumed to be equal to the consumption of the respective costlier state, i.e. when switching from sleep to receive, the cost during the 518 μs is equal to the same amount of time spent in the receive state. During the switch however, the node does neither emit or receive anything. To successfully receive a frame, a node must be in the receive state from frame start until frame end. Switching to receive in between a frame is treated as erroneous transmission.

As the simulation scenarios were chosen not to last more than a couple of hours, we decided to renounce on a battery model that takes self-discharge into account. The sum of the energy consumption therefore simply equals the sum of the energy spent in the respective states.

3.1.3 Clock Drift Model

Sensor nodes are equipped with digitally controlled oscillators (DCO units). DCO make use of a crystal quartz oscillator, an electronic circuit which utilizes the resonance of a vibrating crystal to create a signal with a very precise frequency. Quartz oscillator clocks are still prone to a certain impreciseness, as the frequency slightly depends on environmental influences like temperature, vibration, pressure and magnetic fields.
Clock drifts are measured in quartz frequency tolerance θ, the respective values are given in parts per million (ppm), where 1 *ppm* equals an imprecision of 10^{-6} sec/sec. The lower the frequency tolerance, the more precise the clock. A quartz oscillator usually has no more than $10-20$ *ppm*, which roughly sums up to one second a day. The SaRonix quartz oscillator for example claims to feature a frequency tolerance no more than ± 20 *ppm* in the datasheet [41]. We assumed a clock drift of 30 *ppm* as a realistic choice for a sensor network node's hardware clock. The clock drift at the sender and receiver are assumed to be independent. This assumption is pessimistic, as the quartz inaccuracy is often related to the device temperature and aging, two parameters that are likely to be correlated between the nodes in a network.
The behavior of clock drifts is most easily modelled as univariate random walk process. There are different clock models proposed in literature, but sticking to the random walk is quite usual for networking simulations, as discussed quite frequently in the OMNeT [34] mailing list and certainly in other network simulator communities as well.
The mathematical and statistical properties of this clock model actually fits quite well when comparing with measured clock drifts in practice. In each timestep, a clock drifts away from the *real* time by a randomly distributed value. The model assumes that the

node's clocks develop individual clock drifts d_{node} after n seconds from the simulation start in relation to the *real* time. The drift of a node's clock after n seconds therefore calculates as follows:

$$d_{node} = \sum_{i=1}^{n} X_i \quad \text{where} \quad X_i \sim uniform\ [+\theta, -\theta]$$

where θ denotes the oscillator clock drift in ppm per seconds. The node's respective clock drifts are assumed to make one node's estimation for its neighbors sampling pattern more and more inexact with every timestep since the last relative wake-up pattern exchange. Having two nodes A, B, the *relative* drift between the clocks d_{AB} calculates as

$$d_{AB} = \sum_{i=1}^{n} X_{Ai} - \sum_{i=1}^{n} X_{Bi}$$

with mean

$$E(d_{AB}) = E(\sum_{i=1}^{n} X_{Ai} - \sum_{i=1}^{n} X_{Bi}) = \sum_{i=1}^{n} E(X_{Ai}) - \sum_{i=1}^{n} E(X_{Bi}) = 0$$

and variance

$$V(d_{AB}) = V(\sum_{i=1}^{n} X_{Ai} - \sum_{i=1}^{n} X_{Bi}) = V(\sum_{i=1}^{n} X_{Ai}) + V(\sum_{i=1}^{n} X_{Bi}) = 2n\sigma^2$$

Don Percival proposes in [39] a model with the same statistical properties in the first two momentum's (mean and variance), namely a zero-mean and a linearly increasing variance.

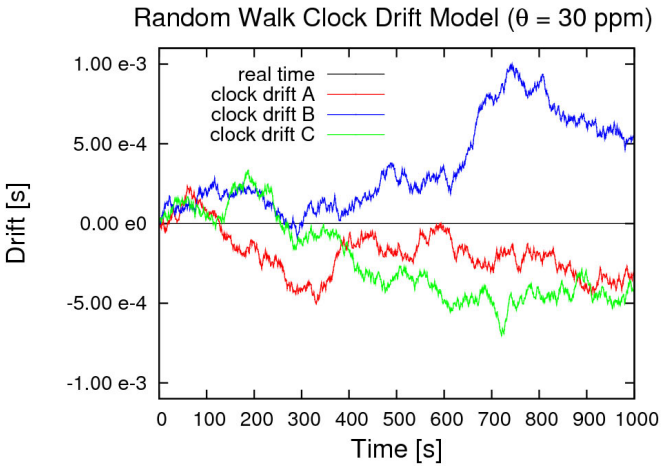


Figure 3.1: Random walk clock drift instances A, B, C compared to the real time

Figure 3.1 illustrates three instances of clocks A, B, C modelled as random walk processes. The difference to the zero-level, which is indicated by the black line, illustrates the drift between a clock and the *real* time. The relative drift between two clocks, as evaluated in the analytical discussion above, corresponds to the difference of two curves.

3.2 Embedded Sensor Boards

Simulation tools are a valuable, manageable and yet cheap test- and playground for evaluating wireless sensor network mechanisms. Simulation tools provide essential insights when developing and evaluating network system mechanisms, yet they are constrained to an artificial horizon. The environment they model is in any case a simplified one. Reasoning over improvements in wireless MAC mechanism design and simulating efficiency gains is much easier than realizing and proving them on hardware prototypes.

The simulation described in 3.1 is an attempt to model a wireless sensor network, with respect to effects of signal dispersion, environmental noise, bandwidth limitation, energy constraints, clock drifts and much more. Yet, many other aspects that may play a role for wireless sensor networks are still left aside. The inherently unreliable wireless channel for instance is not accounted for appropriately. Problems with multipath dispersion, refraction and scattering are not modelled at all. Interferences with other devices (i.e. GSM cellphones), with electromagnetic noise in general and many other issues are untouched. Only measurements on real hardware make sure that all influences and side effects are taken into account. In order to examine the serviceability and robustness of simulated wireless sensor network mechanisms, a prototype implementation on real sensor network hardware is indispensable. We therefore ported the original WiseMAC mechanism and the most promising mechanisms proving substantial efficiency gains on the MAC layer, as well as an on-demand routing scheme to the Embedded Sensor Boards (ESB), a sensor hardware testbed. ESB's as well as the Sensor Node Operating System ScatterWeb have been developed at Freie Universitaet Berlin as part of the ScatterWeb project [43]. The nodes are nowadays shipped by the spin-off company ScatterWeb GmbH [42]. The company subsequently works on improving and updating the tools for application development and programming. The company also distributes the ESB nodes *industry line* which is aimed for use in industry automation, geoinformation systems, logistics and facility management.

The following sections discuss the properties of the hardware testbed, and introduce to the challenges that had to be met with the implementation.
The upcoming chapters discuss results of experiments with simple network scenarios carried out on the simulator and the ESB platform, and address the relationship between simulation results and experiment results.

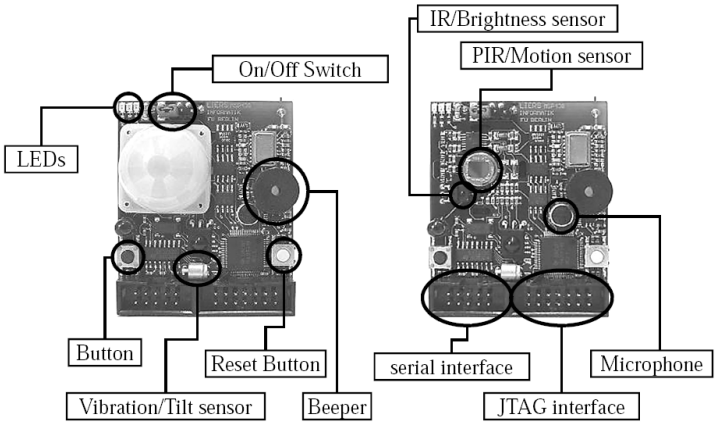


Figure 3.2: Embedded Sensor Board (ESB)

Figure 3.2 shows one of the sensor nodes used. Each ESB is equipped with a micro controller MSP430 and various sensors and communication interfaces:

- 868.35 Mhz Wireless Transceiver
- Infrared sender/receiver
- Motion sensor (passive infrared sensor)
- Luminosity sensor
- Temperature sensor (integrated with Real-time clock)
- Vibration/Tilt sensor
- Microphone/Speaker
- Button

A RS-232 interface allows to connect the node to a personal computer and read and write to the command shell interface of the ScatterWeb Operating System at a rate of 115.2 kbit/s. The parallel interface of the ESB utilizes a JTAG interface (IEEE 1149.1) which allows flashing and real-time program debugging. Monitoring of variables and registers is possible using the debugger of the *MSP430-gcc toolchain* [46].

3.2.1 MSP430 Microcontroller / TR1001 Low Power Transceiver

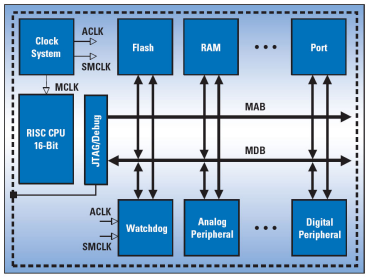


Figure 3.3: Modular architecture of the MSP430 microcontroller

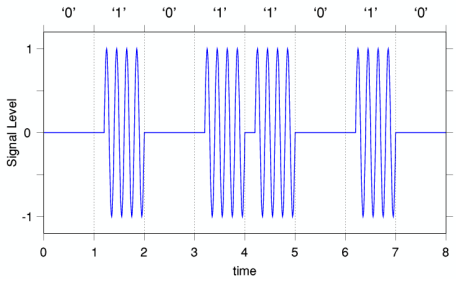


Figure 3.4: On/Off-keyed modulation (OOK)

The microcontroller MSP430 from Texas Instruments [45] is well suited for wireless radio-frequency applications and battery operated scenarios. The 16-bit RISC CPU with only 27 basic instructions is designed for application where low cost and low power consumption are important. The CPU consists in a 16-bit arithmetic logic unit, 16 16-bit internal registers and a control unit. The CPU offers five low power modes LPM0-LPM4 with decreasing tact frequency and energy consumption. Waking up the CPU from the energy saving modes is possible via interrupts.
The 16-bit address space allows to address 60 kbyte + 256 byte flash ROM memory and 2 kbyte RAM. 64 kbyte of EEPROM are available to store large amount of data. In order to communicate with the radio transceiver, one of the two universal synchronous-asynchronous receiver/transmitter (USART) of the MSP430 micro-controller is used. The other USART is used for the RS232 interface.
The MSP430 comes with a sophisticated watchdog solution to recover from program crashes and endless loops or other logical errors. The watchdog consists in a timer and control register. If everything works fine, the operating system periodically updates timer

and control register of the watchdog. If this is not done within a certain predefined time interval, the watchdog automatically resets the CPU. This typically happens when a program or the OS crashes, is locked or got stuck in an endless loop.

ESB nodes are equipped with RFM TR1001 [44] wireless radio transceiver chips. The TR1001 hybrid radio transceiver is designed for short-range wireless communication, and is suitable for applications where small size, low power consumption and low cost are desired. The transmitter includes provisions for both on-off keyed (OOK) and amplitude-shift keyed (ASK) modulation and operates in the license free band at 868.15-868.55 MHz. The ScatterWeb OS per default uses OOK modulation at 19'200 kbps.

OOK is often referred to as simplest digital modulation scheme with very low implementation costs [50]. OOK simply turns the signal on and off for bits to send '1' and '0', respectively. Between each of the data pulses, transmitter output is turned off completely, which explains and justifies the use of OOK in low-power transceivers where conserving power is important. The disadvantage of OOK arises in the presence of an undesired signal, where noise is often misinterpreted as '1'.
The transmission range of the TR1001 heavily depends on the environment. On a plain field without interferences, transmission within a range of up to 300 m is possible. Inside buildings, obstacles like concrete walls and furniture have a vastly reduce the transmission range. Reflection, refraction, multipath propagation, and echo effects decrease the range where nodes are reachable with a sufficient reliability and no bit errors. Nodes are typically not reachable beyond 50 m in buildings. Devices interferring with the transceiver (i.e. IEEE 802.11 devices, GSM cellphones, electronic devices in general) can decrease the transmission range or can temporarily tamper ongoing transmissions, resulting in packets that are not received or received with errors in the checksum.
Power consumption is much lower in the sleep mode than in receive and transmit modes. Almost all wireless radio devices for the input current are listed in section 3.1.2. Further technical information can be found on the technical datasheet of the manufacturer Radio Frequency Monolithics [44].

3.2.2 Power Consumption

The Embedded Sensor Boards integrated circuit aims for least possible power consumption. The ESB is intended to be run with 3 AA-batteries or an external power source, like a mains adapter, a solar panel or a capacitor. The input voltage has to be in the range of 3-5 V. The embedded voltage controller of the ESB then tailors the input voltage to $3V$. The different sensors and the communication interfaces can be turned on and off. Depending on the operation mode of the sensors and the microcontroller, the ESB nodes have different energy consumption levels:

- Average power consumption for the ESB running with all communication interfaces is 45 mW.
- When all sensors are turned off and the TR1001 transceiver module ESB is transmitting data, power consumption is 29 mW in average.
- With every sensor turned off and the CPU in power down mode, power consumption does not exceed 24 μW. Unfortunately, the deep sleep mode is not yet supported by ScatterWeb OS. Up to today it only supports the LPM1 low power mode.
- With all sensors shut off and radio in sleep mode and the CPU in LPM1, the ESB still consumes roughly 14 mW.

3.2.3 ScatterWeb Sensor Operating System

The ScatterWeb Operating System is a very small and simple sensor operating system, consisting in $\sim$ 7000 lines of C code. It uses static memory allocation and provides no abstraction for large amounts of data on external devices, such as a file system.
ScatterWeb is well documented and the source is open to a large extend. In contrast to desktop computer operating systems, ScatterWeb OS is not segmented into core and application. System and applications share the same address space, there's no separation into privileged operating system mode and user mode. All code files are compiled into one single ELF binary file and then loaded to the node's ROM memory.
The control flow of the ScatterWeb OS core is a quite simple never-ending loop function named *superloop* that calls handler-functions if there is something to do and goes to the sleep mode LPM1 if not. The core is contained in the source file *ScatterWeb.System.c* and summarizes how the ScatterWeb OS mainly works. It contains the *main()*-function which is executed when booting the node. After some hardware initialization steps, variable and buffer initializations, ScatterWeb OS straightly heads to the so-called *superloop*, which is listed in Figure 3.5.

```
for(;;) {
  System_startWatchdog();
  // Starts & resets the watchdog, long procedures should call
  // stop watchdog, else MSP will reset.
  if(runModule & MF_SCOS) Threading_eventHandler();
  // Radio tasks.
  if(runModule & MF_RADIO_RX) Net_rxHandler();
  if(runModule & MF_RADIO_TX) Net_txHandler();
  if(runModule & MF_TIMER) Timers_eventHandler();
  // Callback for serial line
  if(runModule & MF_SERIAL_RX) {
    extern volatile UINT8* serial_line;
    if(serial_line != 0) {
      if(callbacks[C_SERIAL]) callbacks[C_SERIAL]((void*)serial_line);
    }
  }
  if(runModule & MF_SENSORS)
  if(Data_sensorFlags != 0x00) Data_sensorHandler();
  if(runModule & MF_RC5) Data_RC5ReceiveHandler();
  dint();
  if(runModule==0) {
    System_stopWatchdog();
    eint();
    LPM1;
  } else
  eint();
}
```

Figure 3.5: ScatterWeb Operating System superloop

The *superloop* contains the *System_startWatchdog()* function call which resets the timer register for the watchdog. If the program gets stuck in a loop during the following statements of the main-loop, the watchdog timer runs out and causes the system to be reset. The variable *runModule* serves as status and flag variable and is used to signalize pending tasks. Some bits of the variable can be set to in an interrupt-service routine, i.e. when a sensor has sensed an event. In such a case, the sensor triggers an interrupt. The interrupt service routine sets the *MF_SCOS*-bit and wakes up the CPU from the sleep mode. The CPU will then execute the main loop and handle the event in *Data_sensorHandler()*.
A very similar procedure is followed for sending and receiving packets, and for handling timer events (in *Net_rxHandler()*, *Net_txHandler()*, *Timers_eventHandler()*). When all handler flags are checked, the watchdog is stopped and the microcontroller goes to the low-power mode (LPM1) again, until the next interrupt starts the main loop.

As the ESB nodes only offer 64 kbyte of memory, thereof only 2 kbyte RAM, the ScatterWeb OS has to get along with a very simple model of concurrency. Real multitasking would consume large parts of the memory resources. Each process would need its own stack and because it in general is hard to know in advance how much stack space a process needs, the space for the stack typically would have to be allocated in advance. A classical multitasked concurrency model would also require complex locking mechanisms to prevent concurrent processes from modifying shared resources.
To provide concurrency without the need for allocating costly stacks for every process and costly locking mechanisms, event-driven systems have been proposed in [48] and [47]. In event-driven systems, processes are implemented as events that run to completion. All events us the same stack, effectively sharing the memory between all processes. Complex locking mechanisms are not necessary. ScatterWeb similarly follows an event-based approach with a simple cooperative scheduler for pending events. In ScatterWeb OS, control heads to the handler function *Threading_eventHandler()* which runs eligible and waiting events in the main loop. Once an event is handled, it is not preempted by other events.
The problem of event-driven operating systems is that long computations can completely monopolize the CPU, making the system unable to respond to external events. This phenomenon can be observed with the ScatterWeb OS. If an event requires some more time to be handled, as costly computations are necessary, the CPU is completely monopolized and all other events have to wait. ScatterWeb *Timers* will not be executed before the computation ends. One will need to periodically reset the watchdog timer register inside the computation code, such that the watchdog does not reset the system before completion of the computation.

3.2.3.1 Transmitting and Receiving

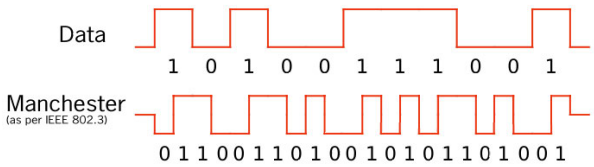


Figure 3.6: Manchester Encoding as applied by ScatterWeb

Transmitting and receiving via TR1001 wireless transceiver is interrupt-driven in ScatterWeb. The two main interrupt service routines *radio_rx_isr()* and *radio_tx_isr()* are called every time the radio transceiver receives or sends one byte in an interrupt-driven handshake. When transmitting, the CPU starts the interrupt loop *radio_tx_isr()* by setting status bits of the TR1001, causing it to turn to the transmit state. In *radio_tx_isr()*, the previously buffered packet contents are read out one byte after the other and written into the TXBUF register of the TR1001. The TR1001 reads the register contents and emits the corresponding signal. It then notices the successful transmission to the CPU by triggering again the *radio_tx_isr()* routine via interrupt. The routine is then called for every byte until the end of the frame is reached. The end of transmission is signalled by setting certain status bits in a TR1001 internal register. When operating with 19'200 kbps, the periodic call to *radio_tx_isr()* takes place approximately every 417 μs.
Receiving is similar. The TR1001 triggers the interrupt service routine *radio_rx_isr()* each time a byte is received. It keeps triggering *radio_rx_isr()* with every byte received, so again every 417 μs. The routine reads out the bytes from the RXBUF register of the TR1001 and buffers the bytes into a receive buffer, continuously calculating the check-

sum. It finishes and sets flag variables when reception is over and saves whether the checksum matches or not. The *radio_rx_isr()* routine then awakes the CPU from the low power mode LPM1. In the *superloop*, the system will notice when reading the *runModule* variable that there has been a frame reception and will interpret the status codes and handle the received frame.

Frame start is determined by a sequence of certain startbytes, frame length by the length indication in the frame header. When starting transmission, a certain well-defined sequence of bytes is being sent in order to announce the upcoming frame transmission. On the receiver side, the received bytes are read out of the RXBUF register and compared to the predefined startbyte sequence. If the startbyte sequence does not match, the node considers the received bytes to be noise. If the sequence of received bytes one after the other matches, frame reception starts and all upcoming received bytes are buffered.

If many '1' or '0' bits are sent in a row, the receiving oscillator can loose the tact and might not receive the bit sequence correctly. Long '1' and long '0' sequences impose a higher probability of transmission errors. In order to avoid long equal bit strings, ScatterWeb logically maps every byte to two bytes with the *Manchester Encoding* algorithm, as depicted in figure 3.6. Manchester Encoding makes sure that long bitseries do not occur. Each bit is encoded as a change in level, rather than a single bit. A '1' is encoded as a low followed by a high ('01'), and a '0' is encoded as a high followed by a low ('10'). It is important to notice that with Manchester Encoding, the transmission rate is halved. The actual transmission rate of the ESB using OOK modulation is reduced from 19'200 to 9'600 bps.

3.2.3.2 Transceiver Switches

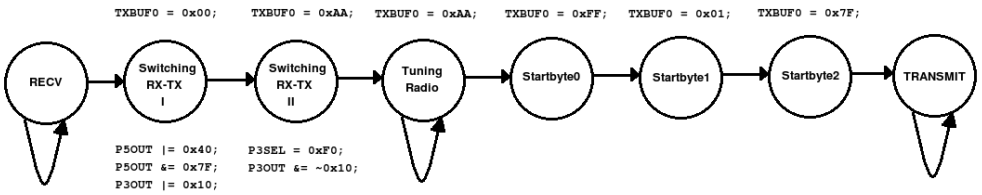


Figure 3.7: Switching from receive (RX) to transmit (TX) in ScatterWeb

When switching from receive state into transmit state, or from sleep state to receive state, certain control-bits of the TR1001 have to be set inside the *radio_rx_isr()* and *radio_tx_isr()* routines. The transceiver itself needs some time to switch to the respective states, as documented in the table 3.1.2, ranging from 10 μs to 518 μs. But in reality, this switch takes much longer. When switching from receive to transmit for instance, *radio_rx_isr()* first sets some status bits, signals transmission start with 0xFF to the USART interface, and then starts tuning the radio with up to 5 preamble bytes and finally begins with the start byte sequence that is necessary for the receiver to detect the frame start. The different steps are illustrated as a finite state machine in figure 3.7. As each isr-cycle takes 417μs, a switch takes about 8 times 417 μs, and adapting the procedure to WiseMAC specific settings led to 3-5 ms turnaround time.
While switching from the receive to the transmit state, it is not possible to listen if the medium is busy. During this vulnerable switching time, another node's frame start can not be detected - collisions can therefore occur even if the medium has been sensed idle before transmission start.

3.2.3.3 Carrier Detection

The TR1001 hybrid transceiver [44] is not equipped with a digital RSSI. Instead, only the last received power *rxpValue* can be read out from the TR1001 internal register ADC12MEM5. This value corresponds to the last measured signal level, and varies heavily between single bits, especially with the simple OOK modulation scheme. Newer transceiver chips calculate digital RSSI values over a sliding window of many measurements. This process is performed on-chip using low noise amplifiers filtering the magnitude of the signal to produce a filtered RSSI indication with low variance.
Because the received power value varies so heavily with the signal modulation, it is not a good indicator for carrier detection. ScatterWeb CSMA therefore only partly makes use of this value and relies the carrier detection merely on a software-based solution. It exploits the reception of the startbyte sequence in the *radio_rx_isr()* routine. When detecting a frame start with the predefined startbytes, the variable *cdCounter* is incremented and the packet that may be destined to another station is overheard. When reaching the frame end, *cdCounter* is set back to zero. In every loop of *radio_tx_isr()*, *cdCounter* is decreased stepwise by one, thereby reaching zero sooner or later. This is a security mechanism that makes sure that the carrier detection sooner or later considers the medium to be free in case a frame start has been detected and the corresponding frame end would somehow have been missed. The variable *cdCounter* therefore serves as carrier detection for the *radio_tx_isr()* routine. If *cdCounter* is greater than zero, the routine considers the carrier to be busy and backs off to avoid collisions. A value of zero indicates that the medium is free.

Yet it is still possible that a frame start is not detected, for instance when a node is turning on its transceiver inside another node's frame transmission. Such a node might attempt to transmit and interfer with the ongoing transmission. To achieve a more powerful and reliable carrier detection, ScatterWeb exploits the received power value of the TR1001. It periodically reads the received power value *rxpValue* out of the transceiver register ADC12MEM5 in a separate interrupt service routine called every 120 μs. In each isr call, *rxpValue* is compared against a maximum receive power threshold *Configuration.rxReceiveLimit* if the variable *cdCounter* has reached zero, what indicates that the node considers the carrier to be free. If the maximum receive power threshold *Configuration.rxReceiveLimit* is exceeded, ScatterWeb increases the *cdCounter* variable and thereby again considers the medium to be busy for at least 13 isr-cycles. The following code snippet in Figure 3.8 illustrates this principle. However, as a single measurement of the value rxpValue is not a good indicator for the carrier state, the periodically called routine *ADC12ISR()* only aims to bridge the gap of unrecognized frame starts and the detection of transmission noise from farther away stations. The mechanism only prevents that nodes imprudently access the medium when the received power is above the security threshold *Configuration.rxReceiveLimit*.

```
interrupt(ADC_VECTOR) ADC12ISR() {
  rxpValue = ADC12MEM5; // Important as this will reset the IFG.
    // Test if CarrierDetect and not already inside a packet.
    if(rxpValue > Configuration.rxReceiveLimit && cdCounter==0) {
      cdCounter = 13;   // Packet start should be detected within next 13 bytes.
    }
}
```

Figure 3.8: Carrier detect in ScatterWeb

3.2.4 Measurement Methodology

Figure 3.9: ESB with 1F-GoldCap capacitor

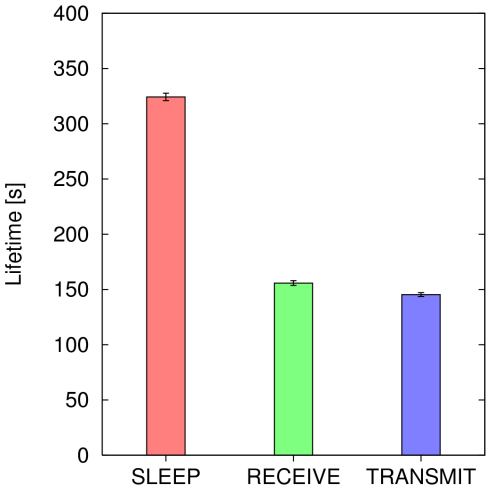


Figure 3.10: ESB operation modes

Measuring the current of a small device such as the Embedded Sensor Boards can be done within some imprecision using a cathode-ray oscilloscope. Such a device is available at the Institute of Applied Physics of the University of Bern. But unfortunately, these oscilloscope devices are not intended to record and sum up the current and the energy consumption over a longer period of time. They are intended to measure the energy consumption in one certain instant.
Equipping all nodes with replaceable or rechargeable AA batteries is not a suitable approach, as measurements of the capacity of customary batteries have shown that the variance can be huge. The capacities of rechargeable batteries that have just been charged up also vary heavily, especially if some of them are new and some have already been used during many charging cycles. It is furthermore too impractical to use batteries or rechargeable batteries to make lifetime and energy-consumption measurements. With energy-saving sensor nodes, the respective lifetimes can last for days or weeks or even months.

We sticked to a a well-tested and established measurement methodology to investigate on the energy consumption of the ESB nodes. The methodology was already applied by the developers of the ESB in [52] and likewise used in the investigation on different MAC protocols in [51]. The methodology uses so-called Gold-Cap Capacitors. GoldCap Capacitors are a special kind of capacitors that come with high capacity of 1 Farad in our case. These devices can be charged quite quickly and power a sensor node for a reasonable amount of time. A 1F GoldCap Capacitor stores up to 15 Joules at a charging voltage of 5.5 V.

The actual absolute value of the nodes' energy consumption is of not much importance in our experiments. We mainly focus on the comparison of different selected mechanisms. The upcoming chapters and sections investigate on the power consumption of different MAC mechanisms using the lifetime of the nodes as an indication for the energy consumption. When being charged with the same initial amount of energy, a node with a lower overall energy consumption can life longer on the energy it is equipped with.

This allows to compare the energy consumption of our own implementation prototype with pure CSMA that does not incorporate any periodic switching between the costly transceiver modes receive and transmit and the energy saving sleep mode. The methodology allows to answer the question how much energy could actually be saved when applying energy-efficiency measures in practice on the Embedded Sensor Boards.

The following methodology setup was applied for all lifetime-measurements, as it produced the most stable measurement results:

- We charged the 1F-Capacitors for a charging time of $\triangle t = 120s$ with a supply voltage of 5.5 V with a customary mains adapter.
- Shutting down all sensors and unplugging the nodes from the RS232 Interface keeps influences and dilutions from other parts low and makes sure that only CPU and transceiver consume energy, besides some small amount of energy spend for the circuits on the board.
- We observed the supply voltage of the capacitor with a customary multimeter. When unplugging the capacitor from the mains adapter, the voltage on the capacitor continuously keeps falling, as sourcing the ESB node with energy slowly discharges the capacitor. We measured the time until the voltage drops below 3 V, which is the supply voltage the embedded voltage controller (MIC5201-3V) requires to power the node. Below this threshold, the node still runs for some small amount of time, but its behavior is unpredictable.

By applying this methodology, we obtained robust and stable results with low variance which allow to compare the ESB-node's energy consumptions in different operation modes. This allowed to quantify the efficiency gains of the energy-efficiency measures under application of different traffic load levels.
In Figure 3.10, the lifetime of a ESB node is depicted, when the transceiver is constantly in one of the three transceiver states sleep, receive and transmit. When comparing the sleep mode with the respective states receive and transmit, it is obvious that approximately one half of the energy charged to the capacitor is being used to power the ESB circuit, microcontroller unit and memory, as a node that is constantly in the sleep mode can live approximately twice as long as a node that is constantly in the receive state.
The lifetime of nodes being constantly in the sleep state gives us upper boundary values for the energy measurements - better results can and will never be obtained when periodically switching the transceiver between sleep and active modes receive and transmit. Similarly, the lifetime measurements of the nodes being constantly in receive and transmit state define the lower boundary values. If the prototype of the power saving MAC mechanisms in the upcoming sections perform worse than $\sim 150s$, they would likewise perform worse than pure CSMA/CA. This would mean that the mechanisms would miss the goal of saving energy. In the worst case this could mean that the energy-saving measures even imply energetic costs, which would prove them to be of no use.

3.2.5 MAC Filter Technique

For developing and testing sensor network mechanisms and analyzing them in multi-hop topologies, it is way too unpractical to deploy nodes in a wide-range scenario during testing and debugging, as recollecting, flashing and redeploying would consume unreasonably much time. We decided to keep all nodes on one big table for all measurements. However, in order to perform measurements of MAC and routing mechanisms in multi-hop topologies, we needed to find a means to control the setup of topology of sensor nodes, to determine which link between which nodes is active and which one is not.
The problem proved to be more difficult than it may seem from a first point of view. The notion of a link in wireless networks in general is not as straightforward as in wired networks. In wired networks, a copper or fibre cable most often corresponds to a link. We have - per definition - no wires in wireless communications. Often, the wireless chip calculates RSSI values to quantify the received signal strength and therefore the quality and robustness of a link. Signal strength thresholds can then be set to determine if a link is used or not.
The fact that the Embedded Sensor Boards' transceiver does not even calculate a digital RSSI did not ease our burden to find a means to control the network topology and link structure. In a first step, we tried to obtain a signal strength indication by saving every value of the received power register *rxpValue* during a packet reception and using a sliding window over the last few values to obtain a filtered, average value. *Buschmann et al.* also experiments with the Embedded Sensor Boards using the received power register *rxpValue* of the TR1001 transceiver in [49]. The investigations conclude that the high variation in the measurements and the only vague correlation with distance and delivery probability do not allow to implement any useful distance estimation.

We therefore omitted to rely the topology control on any measurements of signal power and integrated a MAC filtering technique based on a node's position and a circular virtual range. Like this, we could make sure that the nodes use only the links that are meant to be used. We obtained this quite flexible scheme with the following minimum assumptions:

- All nodes have a position in a 2-dimensional space
- Nodes know their own position (x, y)
- Nodes know their maximum transmission range (MAX_RANGE)

Every node keeps its x and y coordinates in its configuration memory. With every frame and every acknowledgement sent, the sender node appends its x and y coordinates. We used two unsigned 8-bit integers, could therefore set values from 0 to 127 for each coordinate. A node receiving a frame reads out the coordinates of the sender, and determines the distance to the sender node on the MAC layer. If the difference exceeds the virtual maximum transmission range determined by MAX_RANGE, the packet is neglected and not passed to the routing or application layer. Packets may still be received outside the range, but are already filtered out on the level of the interrupt service routine *radio_rx_isr()* and not passed further.
We set the range to 10 in all our experiments. The unit has no meaning in this context, as it only serves to accomplish the MAC filter. Without the MAC filtering technique, keeping all nodes on one table would yield a fully-meshed network topology. With MAX_RANGE being limited to 10, a node's transmission range is limited to a circular field with a radius of 10. Receivers are permitted to have a position with euclidean distance of at maximum 10 units.

Another solution for the same problem would have been to rely the MAC filtering technique on the id's of the neighboring nodes. One could determine for every node which id's are allowed to communicate with it and which are not, supposedly hard coded in the source code that is flashed to the node's ROM.
We find that our solution is more convenient and fits the ad-hoc nature of sensor networks quite well. With our MAC filter approach, nodes still find each other autonomously, and changes in the topology do not necessitate to alter the source code. The flexible scheme permitted us to carry out different multi-hop experiments.

Figure 3.2.5 illustrates the mechanism. Node 4 is located at position $(0, 0)$ and broadcasts a packet to its neighborhood. Node 7 at position $(10, 0)$ and node 8 at position $(10, 10)$ receive the frame and calculate the range to the transmitting node. Node 4 calculates

$$dist = |(0, 0) - (10, 0)| = \sqrt{(0 - 10)^2 + (0 - 0)^2} = 10$$

and *accepts* the frame as $dist \leq MAX\_RANGE$. Transmissions between node 4 and 7 are possible as they are in each others virtual range. On the other hand, node 8 at position $(10, 10)$ calculates

$$dist = |(0, 0) - (10, 10)| = \sqrt{(0 - 10)^2 + (0 - 10)^2} = 14.14$$

and notices that this packet's sender, node 4, is out of its range and *rejects* the frame. Similarly, node 4 likewise rejects any message originated by node 8.

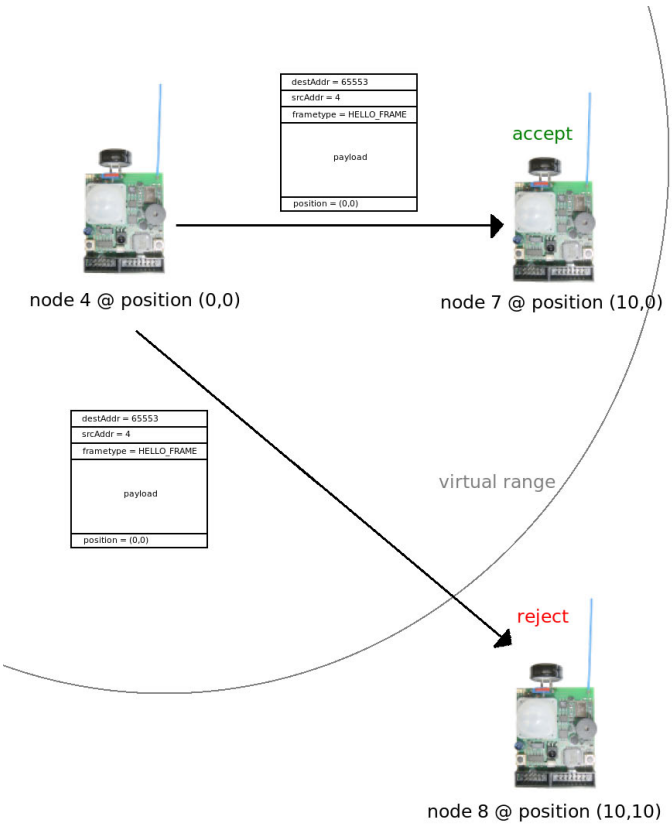


Figure 3.11: MAC filter implementation using a virtual range

3.3 Comparisons between Simulation Results and Sensor Testbed Results

In the upcoming chapters, we cross-compare measurements performed on the ESB with measurements carried out in the simulator environment. In order to obtain a realistic model of the ESB nodes that reproduces similar results in simulation, we had to adjust several model parameters.
Transceiver operation mode switches need more time with the ScatterWeb platform than outlined in the technical datasheet of the manufacturer. To switch from receive to transmit, ScatterWeb needs to go through different steps, as discussed in section 3.2.3.2. The procedure requires roughly 4 ms, whereas the datasheet of the transceiver only accounts for 12 μs. Similarly, switches from transmit to receive and from sleep to receive need more time than indicated in the datasheet. We estimated these switches with 2 ms and 1 ms, respectively.
The power consumption of the ESB in the respective states is modelled according to the measurements on the ESB prototype implementation performed in section 3.2.4. When applying the parameters below, the whole ESB node's power consumption, including CPU, board circuit and memory is taken into account. As measured in 4.2, receiving and sending is more or less equal expensive. The sleep state is still the most energy-efficient, but the difference is not as huge as modelled with the datasheet parameters of section 3.1. To obtain a more realistic energy model, the sleep current is 2.0 mA, receive 4.5 mA and transmit 5.0 mA.
The bit rate was adapted from 19'200 to 9'600 bps, as ScatterWeb applies the Manchester Encoding algorithm to ensure more reliable communication and avert bit-errors.

The simulation parameters for all cross-comparisons are listed in the table below. We will refer to them as the *comparison* model parameters in the upcoming sections. If not explicitly mentioned, the parameters of section 3.1 and 4.1 were used to model the transceiver current, transmission rate and the node's power consumption.

comparison model parameters	
bit rate	9'600 *bps*
minimum preamble	5 ms
medium reservation preamble	uniform [0,6] ms
packet queue length	5
transceiver transition delays:	
recv to send	4 ms
send to recv	2 ms
sleep to recv	1 ms
transceiver current:	
send	5.0 mA
recv	4.5 mA
sleep	2.0 mA
battery (lifetime measurements)	20 J

Table 3.1: Parameters for cross-comparisons of experimentation testbed results

Chapter 4

Implementation and Evaluation of WiseMAC

4.1 WiseMAC in the OMNeT Simulator

simulation parameters	
wake interval duration T	250 ms
wake ratio	5%
bit rate	19,2 $Kbps$
minimum preamble	1 ms
medium reservation preamble	uniform [0,2] ms
packet queue length	15

We modelled the WiseMAC protocol [4] with the preamble sampling technique, the extended carrier sensing range and all the main features of the protocol in the OMNeT simulator to experiment with modifications and optimizations on the unsynchronized MAC scheme.
The implementation in OMNeT is straightforward. Switching between wake and sleep intervals is modelled with *self-messages* telling the node to turn the transceiver to another state. Similarly, the state transition delays of the transceiver are also modelled with *self-messages* being triggered when the switch is initiated, and notifying the node when switching is complete. Preambles are modelled as special MAC packets with varying size. When receiving preambles, a certain flag is set such that nodes do not turn to the sleep mode when the next sleep timer runs out.
The WiseMAC implementation applies a periodic preamble sampling period of $T = 250\ ms$ and a 5% duty cycle. The duration of a preamble sampling interval therefore comes to 12.5 ms. As we later found out with the ESB WiseMAC implementation, even lower duty cycles are possible on sensor hardware testbeds. But when considering the impreciseness due to implementation specific issues and the unpredictable behavior of the transceiver's state transitions, we considered that 5% should be an appropriate and realistic choice.
Figure 4.1 depicts six nodes operating with the WiseMAC protocol in OMNeT. The black circles depict the nodes' respective maximum transmission range. This only serves to visualize the neighborhood a node can reach. The signal propagation model does - as discussed in 3.1.1 - not rely on a simple *unit disk graph*.

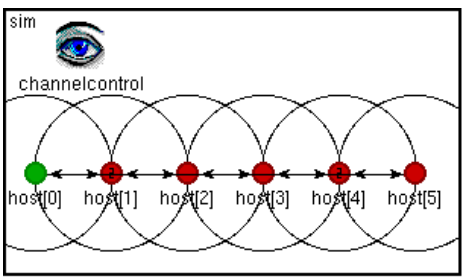


Figure 4.1: WiseMAC nodes in OMNeT

4.2 WiseMAC on the Embedded Sensor Boards

The implementation of the power saving WiseMAC protocol on the ESB proved to be a challenging task. The main features of the WiseMAC proposal outlined in [4] could be realized, some had to be omitted. The parameters listed below led to stable and quite robust functioning of the prototype implementation on the ESB. They were applied throughout all the examinations and measurements of WiseMAC on the ESB and the modifications concerning the wake pattern suggested in chapter 5.

ESB prototype parameters	
basic interval duration T	500 ms
wake ratio	1%
retries	3
minimum preamble	5 ms
medium reservation preamble	uniform [0,6] ms
baud rate	19'200 bps
bit rate	9'600 bps
MAC header	104 bit
payload	96 bit
packet queue length	5

4.2.1 Preamble Sampling and Frame Reception

The WiseMAC sleep-wake pattern is accomplished with ScatterWeb timers calling themselves and other timers in an endless recursion. After boot-up and hardware initialization, a ScatterWeb timer function *tactTimer()* is scheduled. The timer schedules itself again for execution after the basic interval duration T. It continues turning on the transceiver and scheduling a timer to switch the transceiver off again, thereby accomplishing the alternation between receive and sleep state with the basic interval T.
As the transceiver switches need a certain turnaround time, and carrier detection is bound to the recognition of a sequence of predefined startbytes, nodes need a certain minimum duty cycle to actually recognize if a preamble is being sent. The wake-ratio could therefore be lowered down to the minimum value of 1% for $T = 500\ ms$, but not further. The duration of the wake duty cycle calculates as $\triangle t = T \cdot wakeratio = 5\ ms$. In fact, 5 ms is only the time from the moment when the periodic *tactTimer()* writes certain control bits into a transceiver register, thereby telling the transceiver to turn to the receive state, and the moment when another timer function tells it to go to the sleep state again. The transition delay for changing from sleep state to the wake state has to be subtracted from the duty cycle. The net duty cycle therefore comes to only $3 - 4\ ms$ in each cycle.

When the periodic *tactTimer()* is called, the transceiver is switched to recv and listens for the startbyte sequence. If after the duty cycle $\triangle t = T \cdot wakeratio$, the node has not recognized this sequence, the transceiver is switched to sleep state again by the *shutDown()* timer. The node is then kept asleep until the next *tactTimer()* is executed. If the node recognizes the startbyte sequence within the duty cycle $\triangle t$, the shutdown timer does not switch the transceiver to sleep. The node is kept in the receive state until preamble and frame are correctly received. Inside the interrupt service routine *radio_rx_isr()*, the receiving station determines whether the frame is a broadcast or unicast frame. In the first case, the transceiver is ultimately turned to the sleep state. In the latter case, a 10-byte acknowledgement is sent after frame reception. The gap in between

frame and acknowledgement is determined by the transceiver switches and the mechanism for frame reception. In total, the gap roughly takes 10 ms, including the startbyte sequence, whereafter the node is subsequently turned to the sleep state again.

4.2.2 Frame Transmission

In the sender's case, the application passes a packet to the ScatterWeb OS function *Net_send()*. The function prepares the frame in the corresponding buffer and returns immediately (nonblocking call). The *Net_txHandler()* function is then called in the next OS superloop. The function determines if the frame receiver is already known - meaning if the node's schedule offset is already stored in the WiseMAC neighbor table. If this is the case, the node calculates the necessary preamble duration according to the WiseMAC equation $T_{preamble} = min(4\theta L, T)$. The node then calculates the exact instant when it has to switch its transceiver to the receive state in order to contend for the medium, switch to transmit and begin sending the preamble, including a randomly determined small medium reservation preamble. Thereafter it schedules a timer to alert itself in this instant, contends for the medium and sends the preamble and the frame in the appropriate instant. If the medium is not free, it turns to sleep again and schedules a timer to alert itself for the next transmission attempt at the neighboring node's next wake-up.
In case the receiver is unknown yet, the preamble duration is set to the basic interval duration T and the transmission is attempted immediately. After successful medium contention, preamble and frame are subsequently transmitted.

4.2.3 Retransmissions

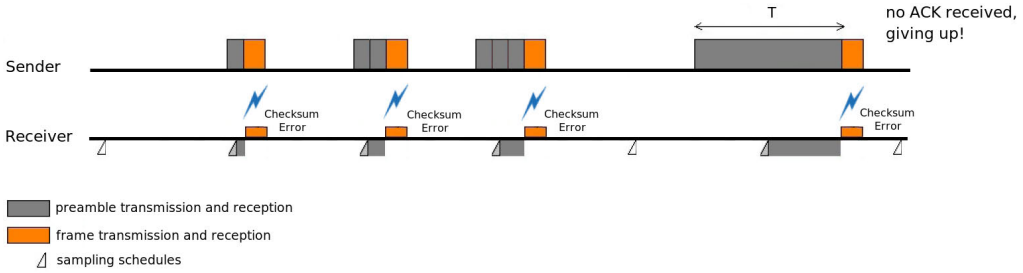


Figure 4.2: Retransmissions in the WiseMAC implementation on the ESB

When a node transmits a packet and does not get a respective acknowledgement within certain predefined time, MAC layer protocols often schedule retransmissions. Sometimes, some parameters are changed in order to increase the probability that the second attempt succeeds. The original WiseMAC draft outlined in [4] does not address the topic retransmission strategy at all. We therefore designed an own solution for retransmissions in our WiseMAC prototype implementation.
A transmission is restrained to four transmissions at maximum, allowing three retransmission attempts. In case the node is not known yet, all transmissions prepend full-cycle preambles of duration T. In case the destination is known, the procedure is illustrated in figure 4.2. For the first transmission, the preamble size is chosen according to WiseMAC to be $T_{preamble} = min(4\theta L, T)$. If this attempt fails, may it be because the transmission is interferred by other transmissions or electromagnetic noise in the channel, as indicated in figure 4.2, the station receiving no acknowledgement within ~ 50 ms starts with a second transmission attempt. In the second attempt, the initial preamble is doubled. In the third attempt, the initial preamble is tripled. Finally, in the last attempt, a full-cycle

preamble is prepended to the actual frame transmission.
Using this retransmission strategy, delivery probability for transmissions between two nodes reached 100% for all ESB nodes that are currently available in the Institute of Computer Science of the University of Bern. Cases where two or more transmission attempts occur, but remain quite seldom, given no exterior influences and parallel transmissions between other sender and receiver pairs. The occurrence of retries furthermore proved to be more node-dependent rather than depending on the rate of transmissions. Some nodes seem to work more reliably than others, which are most likely symptoms of the heavy usage and the countless ROM flashing's in the last couple of months.

4.2.4 Extended Carrier Sensing Range

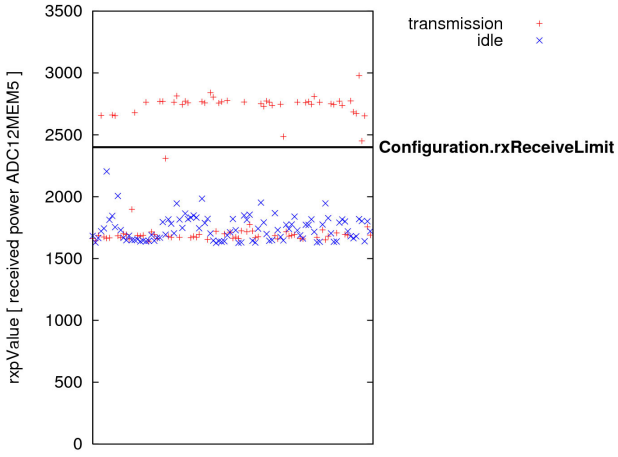


Figure 4.3: Received power measurements during transmission and with idle channel

As discussed in 3.2.3.3, the TR1001 transceiver [44] does not calculate a digital RSSI. Lacking a reliable value for quantifying the signal strength, the implementation of the collision avoidance scheme to mitigate the hidden node problem in WiseMAC, the so-called *extended carrier sensing range* proved to be quite difficult. WiseMAC proposes to apply a lower carrier sense threshold, a threshold that considers the medium to be busy when there is a signal detected, even if the signal is too weak to be correctly demodulated, as discussed in section 2.1.3.
To achieve the effect that a station considers the medium to be busy when it receives transmission noise from farther away stations, we suggest to alter a crucial parameter of the carrier detection mechanism, that is mainly intended to bridge the gap of unrecognized frame starts (see 3.2.3.3). ScatterWeb periodically reads out the received power value *rxpValue* from the transceiver and increases the *cdCounter* if the value is bigger than the threshold value *Configuration.rxReceiveLimit*. It thereby considers the medium to be busy and prevents the transmission handler function from transmitting. Usually, when a transmission is going on in a range of $\sim 10\ m$, the received power value of the TR1001 reaches a value of $\sim 2600 - 2700$ for a '1' and $\sim 1700 - 2000$ for a '0'. Figure 4.3 illustrates 100 measurements of the TR1001 received power value *rxpValue* when a transmission is going on (red dots), and 100 measurements when the channel is idle (blue dots). As one can see in the figure, red dots are on the high level *and* the low level, depending on whether a '1' has just been sensed or a '0'. According to the ScatterWeb UserGuide [42], the threshold value *Configuration.rxReceiveLimit* can be set and varied to figure out optimal settings for a given application scenario. According to comments in the ScatterWeb community, values in between the range of 2300-2400 should suffice.

To achieve the more prohibitive carrier sensing mechanism of WiseMAC, lowering the *Configuration.rxReceiveLimit* threshold value is the only thinkable solution for the ESB sensor nodes. We propose to set the threshold value to 2100 or even lower. The lower the threshold is set, the earlier the node considers the medium to be *busy*. However, setting the threshold value too low will cause the station to consider the medium busy even when no transmission is going on at all. The station will then not dare to transmit any packet anymore.

We neglected to perform measurements and quantify the effect of such a lower threshold when dealing with transmissions from far away stations, as the assumption of circular ranges is unrealistic and simplistic anyway, and such investigations would go far beyond the scope of the thesis. Similar investigations with measurements of the receive power in function of the distance are carried out in [49].

4.2.5 Idle Power Consumption

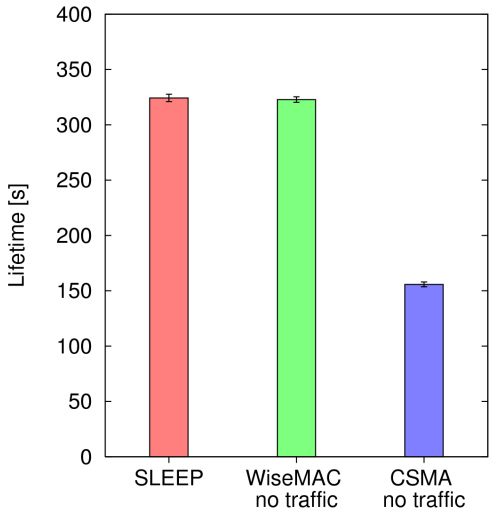


Figure 4.4: sleep mode, WiseMAC and ScatterWeb CSMA (=receive mode)

Figure 4.4 depicts the measured lifetime of an Embedded Sensor Board when applying the methodology described in Section 3.2.4. In Figure 4.4, it becomes clear that the WiseMAC preamble sampling technique is quite energy-efficient. When sampling the medium periodically with $T = 500\ ms$ and a duty cycle of 1%, the cost is barely measurable at all. In case of no traffic, the small medium-samplings lead to a very low power consumption. The lifetime of a node that applies the WiseMAC medium sampling technique (green bar) is almost equal to the lifetime of a node with the permanently turned-off transceiver (red bar).
Considering that the mechanism still allows to keep nodes reachable at any time within 500 ms at maximum, the cost for this connectivity is quite reasonable, almost negligible. When comparing the lifetime of the WiseMAC node to the lifetime of simple ScatterWeb CSMA (blue bar), which keeps the transceiver permanently in the receive state, the lifetime could be increased by approximately 120%.

The WiseMAC protocol implementation on the ESB presented in this thesis certainly is the prototype with the lowest duty cycle and lowest idle power consumption that has up to today been implemented on the ESB research platform.

4.3 WiseMAC Evaluation

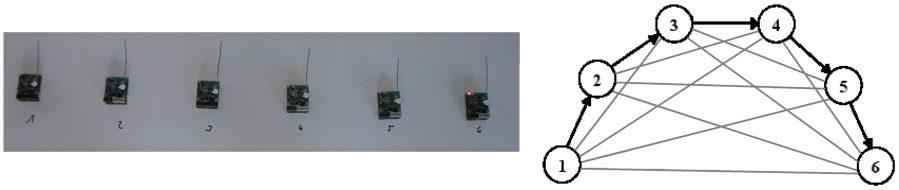


Figure 4.5: Chain topology

This section analyzes the performance of the Embedded Sensor Boards WiseMAC prototype with increasing traffic load and cross-compares the results with corresponding results of measurements with the same experiment setup in the WiseMAC simulation in OMNeT. As a simple scenario setup, we chose a linear chain topology with 6 participating nodes, as depicted figure 4.3. The nodes are all in range of each other, in terms of transmission range. Technically, they build up a full mesh topology, as indicated by the grey lines in figure 4.3, but only the links that are painted in bold are used.
The scenario models periodic traffic along the chain. As all nodes are neighbors, their respective transmissions serves to portray a certain extend of environmental noise and interference by other nodes' transmissions, as it might occur in a dense wireless sensor network topology. With nodes being all in range of each other, increasing traffic will furthermore lead to increasing periodic overhearing. In all lifetime-measurements, we measured the time until the intermediate node 5 depleted.
Before every measurement run, an external node broadcasts a special SYNC packet. Immediately upon reception of the type field in this packet, all nodes reset their clocks back to zero. As this is done inside the receiving interrupt service routine *radio_rx_isr()*, the accuracy of the synchronization among all nodes is expected to lie within one *radio_rx_isr()*-cycle, which is approximately 417 μs. This methodology proved to be sufficiently accurate for our measurements over the five hops.

Node 1 starts generating traffic and addresses all frames to node 2. The application layer in node 1 generates a packet and logs the exact time into it. It then passes the packet to the MAC layer, where it is buffered and sent. Node 2 receives the frame and subsequently forwards all frames to node 3, until the packets reach node 6. When node 6 receives the frame, it passes it to the application layer, where it is decapsuled, and the sending time extracted. Like this, the one-way delay could be calculated as time between the instant when the application layer in node 1 passes the packet to the MAC and the instant when the application layer in node 6 unpacks the frame.

Traffic is generated with varying inter-delay. We aimed to omit effects and results depending on a certain pattern or a certain periodicity that traffic is passed to the MAC layer. The inter-delay between two packets therefore depends on the traffic rate r [frames/sec] and a randomly distributed jittering interval in between two basic wake interval durations T. Traffic should therefore be distributed over the whole basic interval of the first node, such that the effects of the constant alternation between wake and sleep states does not impact on the measurement results. The inter-delay $\triangle t$ between two packets calculates as:

$$\triangle t = \frac{1}{r} + d_{jitter} \qquad d_{jitter} \sim uniform[+T, -T]$$

4.3.1 Lifetime

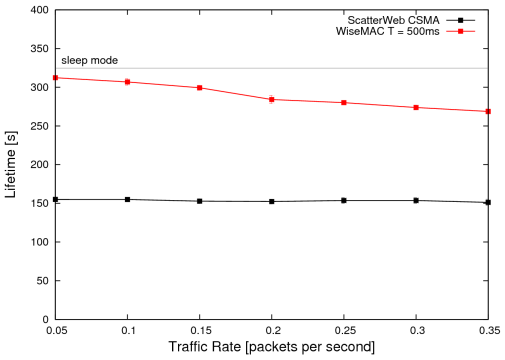


Figure 4.6: Lifetime on the Embedded Sensor Boards

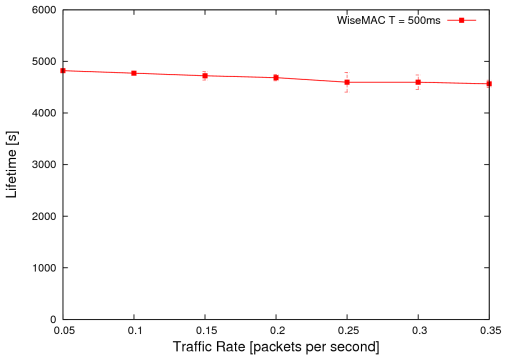


Figure 4.7: Lifetime in the OMNeT simulator

Figure 4.6 on the left depicts the lifetime of the selected ESB node 5 as a function of the traffic rate r when being charged with the initial amount of energy (see section 3.2.4). As the node's energy consumption increases with increasing traffic along the chain, a more or less linear decrease of the node's lifetime can be observed. The black line in the same figure displays the lifetime of a node using custom ScatterWeb CSMA when applying the same traffic. Obviously, the traffic has no greater impact on the curve. ScatterWeb CSMA keeps the transceiver constantly in the receive state, applying no energy-saving measures such as periodic switching between sleep and active states. As sending and receiving is more or less equal expensive, the traffic has no big impact on the lifetime of nodes applying ScatterWeb CSMA.
Figure 4.7 on the right depicts the lifetime of the same node in the simulator. Lifetime is measured as the time a node can live on a certain amount of energy, using the *comparison* parameters of section 3.3. The graph shows a quite similar decrease of lifetime as the graph of the ESB implementation, although the steepness is a bit lower. This might be due to the absence of retries in the simulator and the fact that transmissions including the respective acknowledgements still need a bit longer with the ESB than modelled on the simulator.
These lifetime results on the WiseMAC ESB prototype state clearly that the node's energy consumption can be drastically reduced, whereas considerable service characteristics and the connectivity can still be maintained. With $r = 0.35$, nodes forward a packet along the chain every three seconds. Still, the lifetime is increased by approximately 80% compared to non-power-saving ScatterWeb CSMA. This result is satisfactory and inspiring.

4.3.2 Delivery Rate and Retransmissions

We measured how many of the packets initially sent by the first node finally reached the destination node 6. Astonishingly, there were no packet losses at all. The implementation of the WiseMAC protocol on the ESB managed to deliver every single packet from source node 1 to destination node 6, without a single packet loss.
We further investigated on the count of retransmissions that are necessary to deliver a packet from node 1 to 6. As discussed in 4.2.3, the implementation allows 3 retransmissions, so 4 transmissions in total for each transmission attempt between a sender and receiver. We kept track on the retransmission attempts in a special frame field and summed up all necessary retransmissions for each packet. The retransmits turned out not to depend on the transmission rate for the rates being used above, but to depend

more on the nodes that were used for the test setup. Some nodes inherently delivered worse results than others, which might also be a symptom of the heavy usage during the last years. We measured the retransmissions in separate measurement runs with 500 packets for each run with different nodes. When averaging the measured retransmissions per packet, we obtained values ranging between 0 and 1.61 retransmits per packet transmission from node 1 to node 6. As the results more and more proved to depend on the nodes rather than transmission range or the frame rate, we ceased to investigate further on this topic.

4.3.3 One-Way Delay

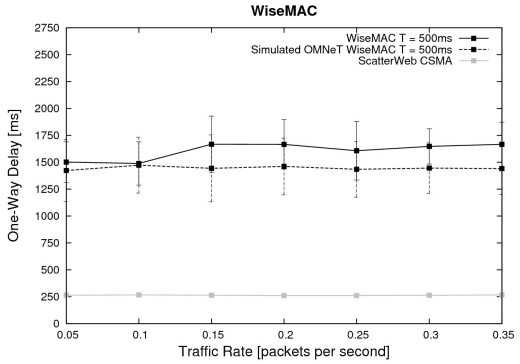


Figure 4.8: One-way delays of WiseMAC in simulation and on the ESB prototype

Figures 4.8 illustrates the end-to-end delay measured both on the ESB prototype and in the OMNeT simulation. As we cross-compare the results with results of the sensor hardware testbed, we again applied the *comparison* parameters. The delay measured is the time difference between the instant when the upper layer passes the payload to the MAC layer of the source node 1 and the instant the destination node 6 passes the payload to its upper layer. The delay proved to be independent from traffic rate. The mean value for the transmission over 5 hops is in the range of about $1500 - 1600\ ms$, approximately 300 ms per hop.
As depicted in figure 4.8, the results of simulation and the ESB implementation fit quite well. The per-hop delay of $\sim 300\ ms$ is obtained both in simulation and on the ESB, and can be explained as follows. If a packet has to be sent from one node to another, the sender node first determines the next wake-up of the receiver node and waits for this instant. As the wake intervals are uniformly distributed over the basic interval duration $T = 500\ ms$, the expected time to wait for the next instant is $E(t_{wait}) = \frac{T}{2} = 250\ ms$. But from the receiver's perspective, the node can not forward the frame to the next node before successful reception and acknowledgement of the frame. It first has to receive the frame, and transmit the respective acknowledgement, and then determine the next wake-up of the next node in line. We can estimate the expected delay per hop $E(d_{hop})$ as the time necessary to wait for the next wake interval of the receiver node plus all delays that are necessary for frame transmission and acknowledgement, i.e. the time for the medium reservation preamble t_{MRP}, the minimum preamble t_{MP}, the transmission of the frame t_{frame}, the transceiver switches t_{rxtx} and t_{txrx} and acknowledgement t_{ack}. As we applied the *comparison* parameters of section 3.3, we can incorporate the expected value $E(t_{MRP}) = 3\ ms$ of the medium reservation preamble t_{MRP}, which is a uniformly distributed value in between [0,6]. When incorporating all particular delays of the transceiver switches of section 3.3 into the calculation, we analytically obtain a per-hop delay of

$$E(d_{hop}) = E(t_{wait}) + E(t_{MRP}) + t_{MP} + t_{frame} + t_{rxtx} + t_{txrx} + t_{ack}$$

$$\sim 250\ ms + 3\ ms + 5\ ms + 20\ ms + 4\ ms + 2\ ms + 10\ ms = 294\ ms$$

With 294 ms analytically calculated per hop delay we obtain a 5-hop delay of 1470 ms, which is approximately the latency measured in the simulation. The slightly higher value of the one-way delay in OMNeT compared to the result of the implementation on the ESB is most likely explained by the fact that a transmission on the ESB takes still a bit longer as the *comparison* parameters actually model it in OMNeT. Especially the delay in between frame transmission and acknowledgement is longer than the transceiver switches, as the implementation needs to first prepare and buffer the acknowledgement and go through some ScatterWeb implementation specific steps in the interrupt service routine. Some other implementation-specific issues may also play a role, i.e. the scheduling of a packet transmission was implemented to include a security gap of some milliseconds, such that the sender node has enough time to cautiously check the carrier before accessing it for transmission.

4.3.4 On the Impact of Simulation Parameters

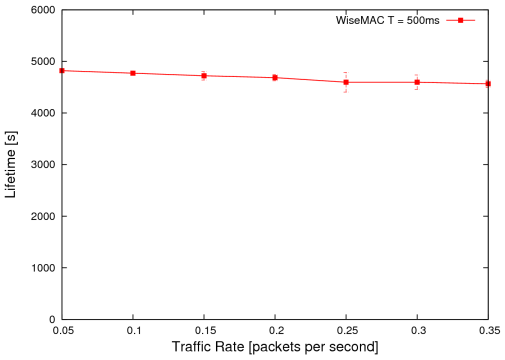


Figure 4.9: Lifetime OMNeT simulator with *comparison* parameters

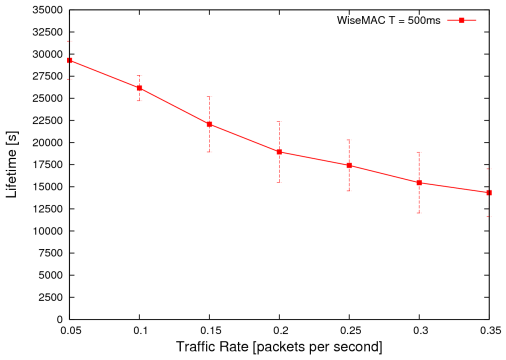


Figure 4.10: Lifetime OMNeT simulator with datasheet parameters

When running the simulation with the transceiver parameters of section 3.1, the resulting one-way delays did slightly differ from the delays obtained with the more realistic *comparison* parameters depicted in figure 4.8, but not that much. The difference in the turnaround delays impacts on the end-to-end delay with approximately $2-3\%$ lower values than the simulations carried out with the *comparison* parameters. To ensure clarity and visibility of the curves, we omitted to display them in the same figure 4.8.

Figures 4.9 and 4.10 display the lifetime curves obtained with the *comparison* parameters and the parameters of the transceiver datasheet outlined in section 3.1. One can clearly see the astonishing impact of the adaptation of the parameters. In the left figure, the lifetime sinks only slowly when increasing the traffic rate, as sending and receiving is only twice as expensive as the sleep state. On the right, the impact of the traffic applied to the chain is much stronger. With more traffic, nodes need to switch and stay in the costly states receive and transmit longer, which leads to the lifetime curve sinking steeper with increasing traffic. With the parameters of the transceiver datasheet on the right, receive and transmit states are approximately 1000 times costlier than the sleep state. In section 4.2 however, we measured that the ratio between sleep and receive and transmit

is rather 1:2:2, as the CPU still consumes the same energy in each state, and the parameters of section section 3.1 only account for the energy spent by the transceiver unit. The datasheet parameters therefore deliver no reasonable energy model for cross-comparisons between simulation and experiments with the ESB.
Choosing suitable and realistic simulation parameters is important. The gap between simulation results and results carried out on real hardware can be lowered when carefully investigating on the parameters of the hardware testbed and modelling implementation-specific settings. Obtaining a realistic model of the reality in simulation is up to a certain degree possible, at least with small experiments where interference from countless distant stations and scale-effects do not yet play a major role.

Chapter 5

Medium Access Control Layer Issues

In this chapter, we outline modifications and optimizations inspired by *Braun et al.* in [2] and [3] on the WiseMAC protocol, and compare it against the original draft standard. We make suggestions by introducing new mechanisms to improve performance, and evaluate them with the OMNeT network simulation environment, and with measurements performed on the Embedded Sensor Boards WiseMAC prototype. The last experiment of this chapter is an attempt to integrate a simple data aggregation and convergence technique into the MAC.

5.1 Clock Drift Evaluation

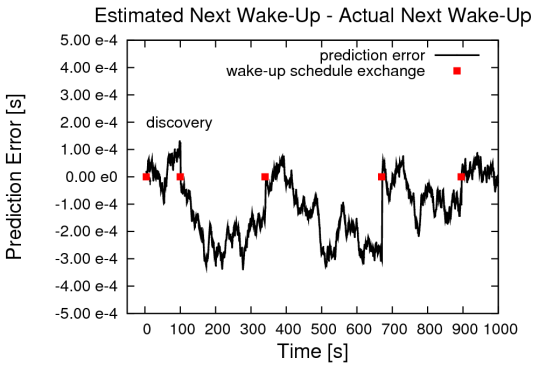


Figure 5.1: Relative clock drift and synchronization between two nodes

Figure 5.1 depicts the simulated relative drift between two arbitrary nodes. The y-axis illustrates the difference in the estimation of node A for the next wake-up of its neighboring node B and its real next wake-up in the WiseMAC simulation, and how this difference behaves when infrequently exchanging messages with the wake-up schedules piggybacked. After $\sim 4s$, node A receives a message from node B, based on which it can estimate the next wake-up of B for the upcoming time.
As WiseMAC renounces on a global synchronization scheme, every node must keep a table with the relative schedule offsets for its neighbors. Neighboring nodes then correct each others relative clock drifts when there are transmissions in between the two neighbors, or if they overhear each others transmissions or acknowledgements with their wake-up schedules appended. When receiving a schedule update, the relative clock drift falls back to zero.
This mechanism is marked with red spots in figure 5.1 after 100 s, 340 s, 670 s and 895 s. Node A then receives or overhears a message from B with its wake-up announcement appended, and updates the relative schedule table for its neighbor B. As consequence, the relative drift to B depicted in 5.1 falls back to zero.
We renounced to investigate on the clock drift behavior on the ESB, as there was no possible methodology available to figure out clock drifts of ESB's with a sufficient accuracy, and such an investigation would be far too time-consuming.

5.2 Preamble Length

WiseMAC suggests to apply a preamble of duration $T_{preamble} = min(4\theta L, T)$, in order to compensate for the maximum drift having occurred between two clocks since the time L of the last schedule exchange. This preamble guarantees to compensate for the drift between two clocks in every case, it incorporates every possible clock drift model based on the assumption that a clock drift has a upper bound θ, and includes the worst-case scenario, that a clock could drift apart by the same θ in *every* timestep. When working with the ESB's, we noticed that the linearly increasing preamble function might overestimate the clock drift between two clocks, and that a shorter preamble would suffice.

In the following we show that the WiseMAC preamble function $T_{preamble} = min(4\theta L, T)$ indeed overestimates the clock drift when modelling this drift by a random walk time series process. One can find an optimal preamble duration by incorporating a probabilistic energy cost model, such that the *expected value* of the transmission costs T_c is minimized. When the time in between two relative synchronizations, the so-called inter-delay, is short, this preamble modification does barely pay off. The benefit rises with increasing inter-delay. We model the energy cost as function of the preamble of length x, the cost weight which is specified by the transmission and receive power consumption c_{tr} and c_{recv}, and the probability that the transmission succeeds when using the preamble of length x. The probability that the transmission succeeds when using a preamble of length x can be expressed as function of the cumulative distribution function of the random walk process after n timesteps, $p = f(cdf(x, n))$.
To calculate the expected transmission costs, we have to distinguish two cases. In the first case, the preamble succeeds in alerting the receiving node. In the second case, the preamble is too short and the receiving node does not capture the full frame and the transmission fails. Let us assume that in this case, the sender node stays in the receive mode for a certain waiting delay period $t_{w_{ACK}}$, then turns back to sleep and awaits the next wake interval of the receiving node. As the probabilistic preamble has failed, it now sends a WiseMAC preamble $T_{preamble} = min(4\theta L, T)$, as this preamble duration suffices for any possible drift. In this case, the transmission costs have to account for the energy spoiled for the first preamble, the second preamble, and the first, lost frame of length l_f. By incorporating the additional overhead in case of the first attempt failing, we obtain the following transmission cost function with probabilities p and $1-p$ for first and second cases:

$$T_c(x, n) = \begin{cases} xc_{tr} + l_f c_{tr} & p \\ xc_{tr} + l_f c_{tr} + t_{w_{ACK}} c_{recv} + min(4\theta n, T)c_{tr} + l_f c_{tr} & 1 - p \end{cases}$$

The expected value of the cost function then yields as

$$E(T_c) = p \cdot \left(xc_{tr} + l_f c_{tr}\right) + (1 - p) \cdot \left(xc_{tr} + l_f c_{tr} + t_{w_{ACK}} c_{recv} + min(4\theta n, T)c_{tr} + l_f c_{tr}\right)$$

It might happen that the receiver is early and the sender late, or vice versa. To incorporate both cases, we have to multiply the upper expression by a factor of 2 to obtain the right preamble size which allows to reach the receiver. We numerically optimized the preamble length x which solves the problem with the minimal expected cost and the simulation parameters of section 3.1 and 4.1.

Figure 5.2 depicts both, the WiseMAC preamble of duration $T_{preamble} = min(4\theta L, T)$ and the energy optimal preamble which was obtained when applying the energy cost

function and numerical optimization described above. As one can see, the difference between both preamble lengths does not differ much for low values of t. With periodic traffic with inter-delay between of some seconds, the energy-optimal preamble does not differ much from the WiseMAC preamble. But when dealing with very low traffic, for instance traffic with inter-delay of 50-100s, the WiseMAC preamble heavily overestimates the clock drift between sender and receiver. The energy optimal preamble is in this case much shorter.

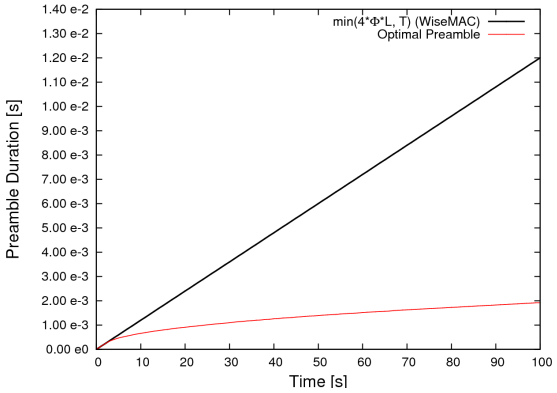


Figure 5.2: WiseMAC preamble and optimized preamble

Simulation Results

We simulated the mechanism in a sensor network scenario where the last node in a chain of 6 nodes sends packets towards the sink, with low rates of traffic. We assumed to achieve increasing benefit out of the optimized preamble when lowering the traffic rate. The table below lists the simulation results for low traffic rates between 0.05 and 0.001 packets per second. The results confirmed the expectations, but the energy efficiency gain is very low. The first column specifies the traffic rate run in the experiment. Second and third column specifies the total energy consumption of all nodes when using the WiseMAC preamble or the optimized preamble, respectively, and fourth column quantifies the efficiency gain of the approach.

Traffic rate λ	Energy WiseMAC preamble	Energy *optimized* preamble	Δ
0.05	18.37 J	18.09 J	1.48%
0.01	11.27 J	11.09 J	1.60%
0.005	12.05 J	11.85 J	1.63%
0.001	10.57 J	10.13 J	4.12%

After having further investigated on the behavior of clock drifts, we found that there are different models with different statistical properties. The so-called optimized preamble only fits to the clock drift model with the random walk, as specified in 3.1.3. It might not be appropriate for other clock drift models. Shortening the preamble and thereby saving some small amount of energy remains questionable, as one risks worsening packet delivery reliability and service characteristics, which might occur with too short preambles.
We renounced to implement this scheme on the ESB nodes. Improvements in the range of 1% would not be measurable and distinguishable from the measurement variation with the methodology of section 3.2.4. As the capacitor can only power the node for approximately 5 minutes, traffic rates of 0.001 can not be examined with this methodology anyway, as the inter-delay between two transmissions would be too long.

5.3 Broadcasting Scheme

Broadcasting is a costly issue in WiseMAC. As the nodes all sample the medium in their own wake-up pattern, a node intending to broadcast a message needs to first alert all its neighboring nodes for the upcoming transmission. The WiseMAC approach consists in prepending a preamble of the duration of the full basic interval duration T to each frame. This broadcasting scheme wastes a lot of energy for sending and receiving the long preamble, whereas the actual data transmission can be short. But regardless of prior knowledge about the neighboring nodes, this mechanism allows to alert every neighbor.

If the message has to be rebroadcast in every node, the characteristic wireless *broadcast storm problem* is certain to occur. The broadcast storm problem for radio networks, as studied in [33], occurs with uncontrolled flooding techniques in wireless networks. If every node rebroadcasts an incoming broadcast message, transmissions take place more or less simultaneously. As a consequence, the medium will be blocked for a long time and many nodes will redundantly rebroadcast the message and receive countless duplicates. Transmissions will presumably collide with other ongoing transmissions, which can in turn lead to starvation of the flood, as broadcasts are unacknowledged and collisions can not be detected by the sender.
When applying no control measures for multi-hop flooding, the WiseMAC broadcast exacerbates the broadcast storm vulnerability with the long preambles. The full-preamble broadcast blocks the channel not only for the immediate neighboring nodes, but also for all nodes in the extended carrier sensing range (as discussed in 2.1.3). Network activity will be restraint to countless full-preamble broadcasts for every rebroadcast, and this is especially costly as all nodes sampling the medium will stay awake when hearing the preamble tone, even if they have yet received the message. As using full-cycle preambles in every node is energetically costly, the designer *El Hoiydi* reflects in [5] that *more sophisticated broadcasting and flooding techniques for multi-hop ad-hoc sensor networks and MANETS remain to be designed*. We aim to bridge this gap with the technique being introduced in this section.
The broadcast storm problem is illustrated below. Figure 5.3 illustrates a node initiating a broadcast flood. The message is received by its neighboring nodes, which all rebroadcast the packet in 5.4. The *black* arrows depict useful transmissions and receptions. As indicated by the *red* arrows, many transmissions and especially receptions are redundant. As receptions are likewise costly in wireless networks, this problem has severe impact on the energy consumption of the participating nodes. If the rebroadcasts of the nodes are not coordinated, frames may presumably collide with each other. The service characteristics of the network can temporarily be harmed.

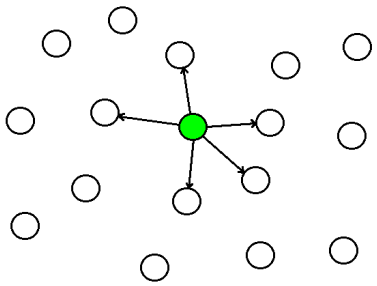


Figure 5.3: Node initiates broadcast flood

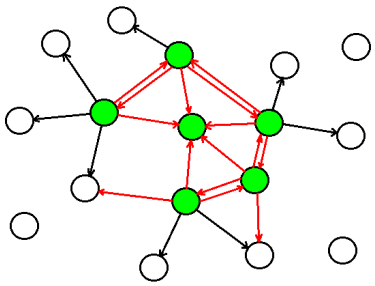


Figure 5.4: Many transmissions and receptions are redundant

5.3.1 (k) Best Instants

Braun et al. study the problem of disseminating broadcasts in a multi-hop topology in [3], when applying the unsynchronized wake-up pattern discussed in [2] (section 2.1.4) that shares a similar wake-up pattern with WiseMAC. The paper concludes with a suggestion for a flooding service that proved robust for use in on-demand routing schemes as well as energy-efficient. The concept is referred to as *k-best-instants* broadcast. This flooding concept is a simple means to limit the broadcasting overhead in power saving mechanisms with asynchronous wake intervals. It suggests to first figure out the minimal set of instants for a packet to be sent with which a node reaches all its neighbors. Having done so, a node shall - depending on required reliability, topology or density of the network - only choose k such instants, where k is meant to be kept small (i.e. k=2). With the factor k, the message *fan-out* can be limited and controlled. Thanks to this restriction, the broadcast storm problem became negligible. Intermediate nodes participating in the flooding do not aim to reach *every* neighbor, but only a limited subset. In contrast to OLSR, which first calculates so-called multi-point relays over its 2-hop neighborhood, the mechanism only relies on the local information about the 1-hop neighbors' wake-ups. The choice of the subset of neighbors therefore is a random one, as it is based on the wake-up patterns, which in turn are determined at random. The flooding technique therefore shares some similarities with probabilistic multi-hop broadcasting techniques.

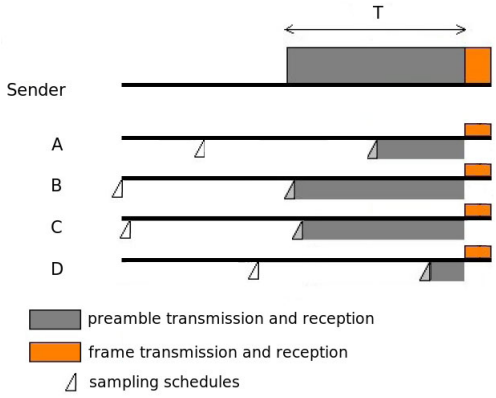


Figure 5.5: WiseMAC broadcast

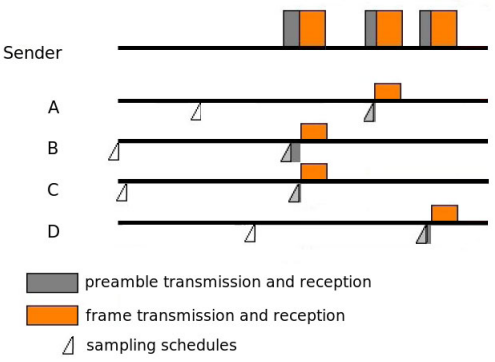


Figure 5.6: k-Best-Instants broadcast

We propose to integrate the same idea to improve the energy-efficiency of the WiseMAC broadcast in general. Consider the situation depicted in figure 5.5. The sender has neighbors A, B, C, D, and already is aware of their individual wake-up schedules. As depicted in 5.6, calculating the best instants for sending a preamble and the frame is much more efficient than using a full-cycle preamble in WiseMAC in figure 5.5. The gray and red area illustrates the time that nodes have to spend in receive or transmit mode.

The procedure to obtain the minimal set of instants for a node to broadcast a message to its neighbors is slightly different than proposed in [3]. It was adapted to the WiseMAC medium access scheme, which consists in only sampling the medium for a minimum amount of time to detect a preamble signal. Finding intersections between such small wake-ups is very improbable. But it is possible to make efficiency gains with an extended concept of the intersections, designed for application with the preamble sampling technique. In contrast to the concept of the intersections of the duty cycles in [3], the mechanism of searching intersections was adapted to consider groups of nodes with *near* wake intervals, such as nodes B and C in figures 5.5 and 5.6.

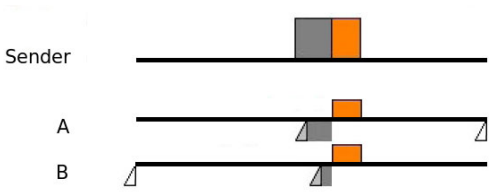


Figure 5.7: Near wake-ups

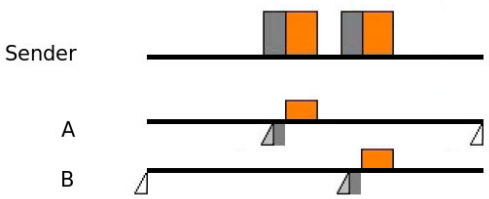


Figure 5.8: Not near enough wake-ups

Wake intervals are considered to be *near* if the difference between their startpoints is smaller than the transmission time it takes to transmit a node's preamble and the actual payload. It might pay off to group *near* wake-up intervals of neighboring nodes and transmit a slightly longer preamble, as sending a preamble and the message twice would be costlier than grouping these two instants and sending the payload only once. We can express the notion of *near* wake intervals analytically as a function of the cycle duration T, the bandwidth b and the size of the frames to be transmitted d - omitting the state transition delays and the cost for the sleep state in this analytical discussion. To be *near* enough, the node with the later wake interval has to be at maximum two halve preambles $\frac{P_A}{2}$, $\frac{P_B}{2}$ and the duration of a packet's transmission $\frac{d}{b}$ away. Two nodes' wake intervals t_A, t_B are *near* if the following condition holds:

$$near(t_A, t_B) := ((t_B - t_A) < \frac{P_A}{2} + \frac{d}{b} + \frac{P_B}{2})$$

Near nodes are depicted in figure 5.7. When the difference between the startpoints of the medium samplings is lower than the actual transmission time for the first node's halve preamble and the second node's halve preamble and the payload, it makes sense to group these nodes and calculate the preamble that is necessary to reach both of the nodes, and yet transmit the payload only once.

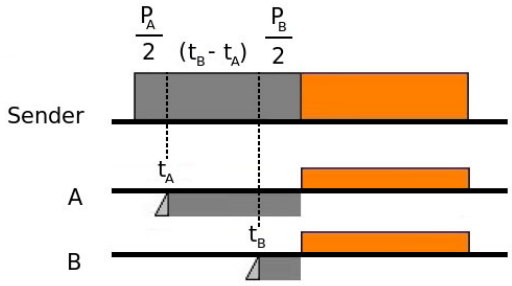


Figure 5.9: Preamble duration when grouping near nodes

In case of *near* nodes, the preamble must suffice to alert both nodes. Let them denote as the sooner node A and the later node B with estimated wake-ups t_A, t_B and WiseMAC preambles P_A, P_B. Figure 5.9 illustrates how the preamble for a group of near nodes is composed. The actual instant when the transmission of the preamble must be scheduled for is

$$t_{group} = t_A - \frac{P_A}{2}$$

The duration of the preamble calculates as

$$P_{group} = \frac{P_A}{2} + (t_B - t_A) + \frac{P_B}{2}$$

Proof: sending a preamble of size P_{group} plus the frame $\frac{d}{b}$ in order to reach two nodes A, B pays off when exactly when their wake intervals are *near*. It pays off if $P_{group} + \frac{d}{b}$ is smaller than sending two preambles P_A, P_B and twice the frame $\frac{d}{b}$.

$$\frac{P_A}{2} + (t_B - t_A) + \frac{P_B}{2} + \frac{d}{b} < P_A + P_B + 2\ \frac{d}{b}$$

simplifying the inequation yields

$$(t_B - t_A) < \frac{P_A}{2} + \frac{P_B}{2} + \frac{d}{b}$$

which is exactly the property $near(t_A, t_B)$ two wake-ups t_A, t_B have to fulfill. □

The algorithm for figuring out the best instants to reach all neighbors that has been used in simulation and on the sensor testbed works as follows:
Given the estimated wake-ups $t_1, t_2, \ldots, t_n$ of all neighbors $N_1, N_2, \ldots, N_n$. Let R denote the set of uncovered instants and T denote the resulting set of instants.

$$R = \bigcup_{i=1}^{n} t_i$$

$$T = \emptyset$$

$**** if\ there\ are\ near\ nodes,\ group\ them\ ****$

$FOR\ each\ pair\ of\ instants\ (t_u, t_v)\ in\ R\ do$
$\quad IF(\ (t_v - t_u) < \frac{P_u}{2} + \frac{d}{b} + \frac{P_v}{2}\ )\ do$
$\quad\quad T = T \cup \{min(t_u, t_v)\}$
$\quad\quad R = R \setminus t_u$
$\quad\quad R = R \setminus t_v$
$\quad ENDIF$
$ENDFOR$

$**** if\ not,\ add\ the\ instants\ separately ****$

$$T = T \cup R$$

T contains the minimal set of instants $t_1, \ldots, t_k$ for broadcasts to be sent to reach all neighbors. The set is sorted and instants with which two *near* nodes can be reached are listed first. The concept can be extended to consider three, four or more *near* nodes, which is however increasingly improbable. The question whether it pays off to apply the algorithm depends on the question how many neighbors there are in the neighborhood to be alerted and how many of them ultimately have to receive the message. For the sender, it pays off to apply the technique if the cost of all preambles $P_1, \ldots, P_k$ and all transmissions for the respective instants $t_1, \ldots, t_k$, denoted as $c_{best-instants}$, is less than the respective cost for one WiseMAC full-preamble broadcast. This can be expressed as

$$c_{best-instants} = \sum_{i=1}^{k} P_{t_i} + (k\ \frac{d}{b})$$

whereas $c_{full-preamble-broadcast}$ can be expressed as

$$c_{full-preamble-broadcast} = P_T + \frac{d}{b}$$

For the sender, it pays off to apply the best-instants technique if

$$c_{full-preamble-broadcast} > c_{best-instants}$$

In the case of the receiver, the best-instants technique pays off in any case, as listening to a full-cycle preamble is costlier than listening to a preamble that only serves to compensate for the individual clock drift.

The multi-hop flooding technique of the k-best-instants as discussed in [3] now proposes to choose the first k elements of this sorted set and transmit the packet at the particular instant. If the number of nodes n is smaller than k, every node will be reached. If $n > k$ and no intersections can be exploited, the node reaches only a subset of its neighbors. There are cases where it is advantageous not to reach the whole set of neighbors with each node's rebroadcast, as the broadcast storm problem [33] can lead to bad performance in radio-channel networks applying uncontrolled flooding techniques. With the parameter k, the network designer can control the count of rebroadcasts, and has a means to control and circumvent the occurrence of the broadcast storm problem.

In the following, we outline two experiments to measure and quantify the performance gain of the (k)-best-instants broadcasting technique. In the first experiment, we measure the lifetime of a node broadcasting messages with a certain rate to its neighboring nodes when applying either the WiseMAC full-preamble broadcast or the best-instants broadcast, but not limiting the number of neighbors that are reached with a broadcast. In the second experiment, we test out the performance in an AODV route request scenario. We simulate and measure how much energy can be saved within all nodes when each node establishes a route to the sink by initiating an RREQ-RREP route discovery cycle.

5.3.2 Lifetime Experiment

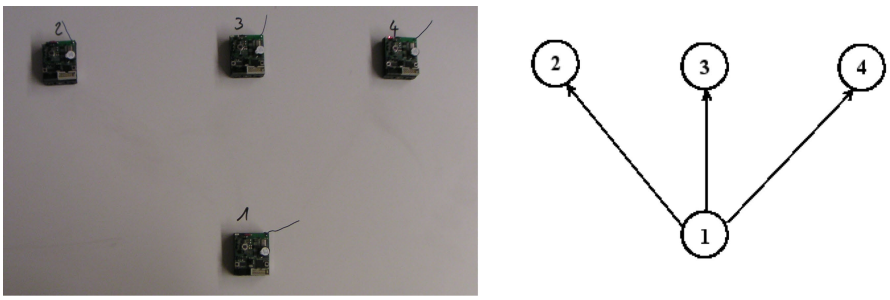


Figure 5.10: Broadcasting scenario

The experiment setup is very simple. Node 1 broadcasts frames to its three neighboring nodes $2, 3$ and 4, as outlined in Figure 5.10. By increasing the rate of broadcasts, we aim to analyze how the rate of packet transmissions impacts on the power consumption. We run the experiments with both broadcasting techniques, the original WiseMAC broadcast prepending a full-cycle T preamble to each frame, and the best-instants-technique. In order to make reasonable comparisons, the best-instant broadcast is not restraint to a certain amount of transmits, and considers every node's wake interval. Both broadcasting techniques therefore aim to reach all three neighboring nodes. We applied the *comparison* parameters for this simulation. The values are listed in section 3.3 and are tailored to model the ESB as close as possible.

Figure 5.11 depicts the lifetime of the broadcasting node as a function of the broadcast rate measured on the ESB prototype platform. Figure 5.12 illustrates the lifetime of a broadcasting node on the OMNeT simulator when being supplied with an initial amount of energy. As expected, lifetime sinks when increasing the rate of transmitted frames. Both figures show that the best-instants technique leads to a limited performance improvement. The measured lifetimes on the ESB prototype are only very slightly longer, and do not exceed 5% in any of the traffic situations. On the simulator, the picture is a bit clearer. As expected, the gap between the both techniques becomes bigger with increasing broadcasting rate. The more the broadcast is being used, the more it pays off to use the best-instants technique.

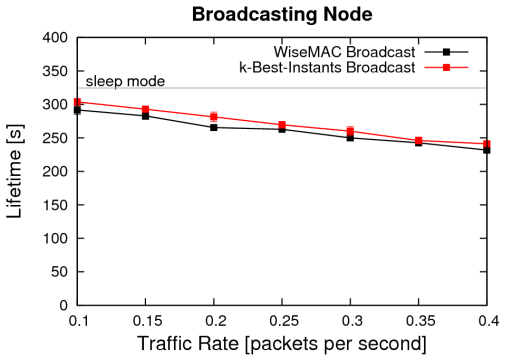


Figure 5.11: Sending node using both broadcast techniques on the ESB

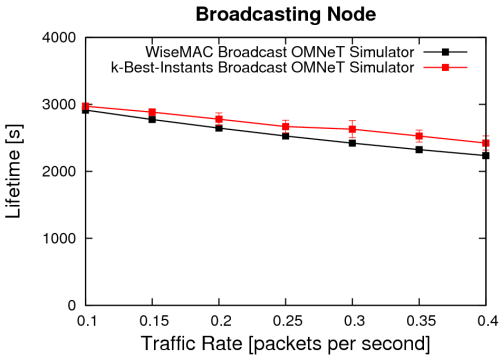


Figure 5.12: Sending node using both broadcast techniques in simulation

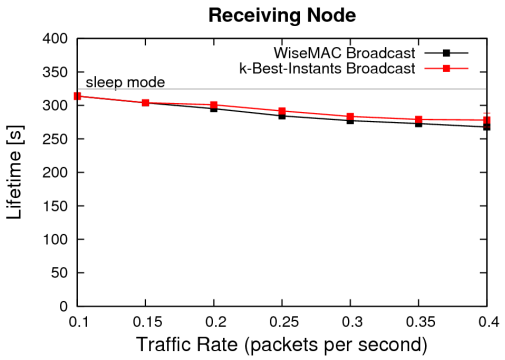


Figure 5.13: Receiving node using both broadcast techniques on the ESB

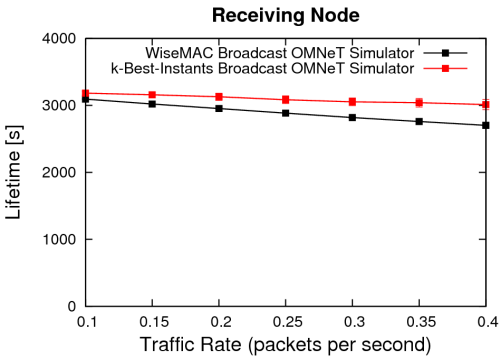


Figure 5.14: Receiving node using both broadcast techniques in simulation

Figures 5.13 and 5.14 depict the measurements when running the same scenario but equipping the receiving node 2 with an initial amount of energy. The performance gain of the best-instants-technique should not only concern the sender node, but also the receiving nodes, as listening to a very long preamble is much costlier than only listening to short preambles. As figures 5.5 and 5.6 illustrate, sending short preambles and the payload several times is especially advantageous for the receivers.
The results confirm the performance improvement in both cases, but again the net gain is very small. The net gain does not exceed 5% in any of the lifetime measurements pursued on the ESB platform. The results of the simulator state similar results, but the gap becomes clearly visible with increasing broadcasting rate. The performance gain reaches approximately 10% with the rate of 0.4 frames per second.

Still, when dealing with only a few neighboring nodes, the approach of the best instants makes sense. If the neighborhood is known and does not change frequently, there is no need to use full-preamble broadcasts. One has to keep in mind that the performance gain on the receiver's side concerns *every* neighbor covered. Not only node 2 has a slightly lesser consumption when applying the best-instants-technique, but also node 3 and node 4. Sending long preambles of the full cycle should be avoided whenever it does not serve to discover or rediscover neighboring node and their respective wake-patterns. Whenever there is a limited amount of neighboring nodes, the best-instants technique should be applied to save energy both at the sender and at the receiver's side.

Furthermore, the best-instants-scheme is more suitable for broadcasting over multiple hops, as the channel will be occupied only for short transmissions. As WiseMAC applies the extended carrier sensing range concept for the collision avoidance, every transmission blocks the medium not only for stations in the transmission range, but also for stations in the larger carrier sensing range. The impact of sending long preambles of duration T is therefore much bigger, in respect to energy wastage and throughput limitation. Every node in the larger range will sense the tone have to stay in the receive mode.

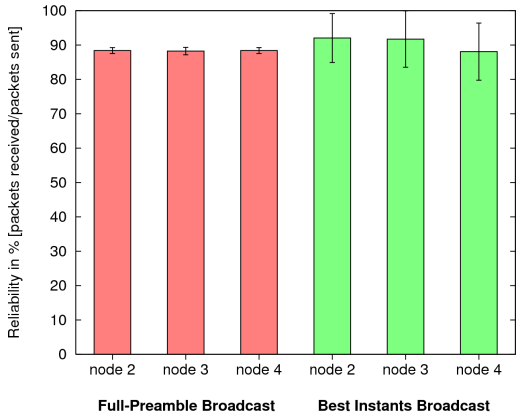


Figure 5.15: Reliability of both broadcasting techniques on the ESB

Figure 5.15 depicts the reliability of the broadcast techniques implementations on the Embedded Sensor Boards. The three bars correspond to the receiver nodes $2, 3, 4$ in figure 5.10 and illustrate how many of the packets that were sent by the broadcasting node actually were successfully received. The red bars correspond to the original WiseMAC full-preamble technique, where for every packet, a preamble of duration T is prepended. The green bar illustrates the reliability for the best-instants technique. The reliability proved to be independent from the rate of broadcasts. It reached approximately 90% with both techniques, with a bigger variance for the best-instants technique approach.

As we did not incorporate another error model in our simulation other than transmission errors caused by other nodes' interference, investigating on the reliability in the simulator in the same scenario yields a rate of 100% for both approaches. As there is only one sender, there is no interference, and every transmission reaches the receiver as intended.

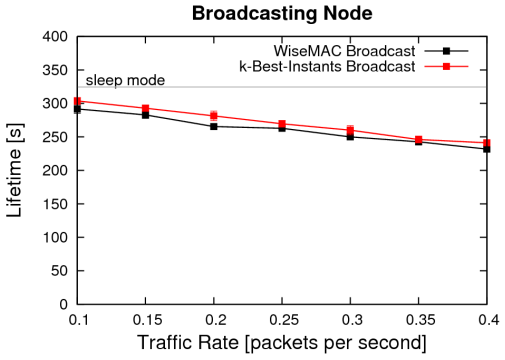


Figure 5.16: Both broadcasting techniques on the Embedded Sensor Boards

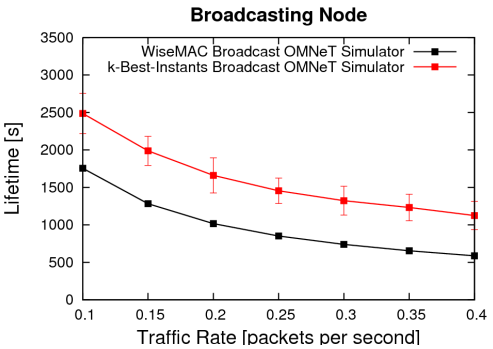


Figure 5.17: Both broadcast techniques in simulation with the datasheet parameters

Figure 5.16 again depicts the results of the same experiment carried out on the ESB, whereas 5.17 depicts the results of the same simulation setup, but with the simulation parameters chosen according to the transceiver datasheet, as listed in section 3.1.2. One instantly notices the much bigger gap between the graph of the best-instants technique and the WiseMAC broadcast when comparing both figures, and reasons why the technique pays off that much better with the datasheet parameters. The reason lies in the different ratios of the parameters for the operation modes. The parameters applied in figure 5.17 assume an energy consumption ratio of approximately $\frac{1}{1000} : 1 : 2$ for the states sleep:receive:transmit, whereas the measurements in section 4.2.5 yield different ratios. Saving some amount of time in one of the costly states receive and transmit likewise pays off by a much longer lifetime. With a rate of 0.4, the lifetime is almost twice as long with the best-instants technique.

Obviously, these parameters do not appropriately model the energy consumption of the ESB nodes in their respective states. They only account for the energy spent by the transceiver unit and deliver no reasonable energy model for cross-comparisons between simulation and real sensor experiment result, at least not for the Embedded Sensor Boards. With the ESB nodes, the sleep mode is indeed cheapest, but as the CPU still consumes the same amount of energy, an ESB in the sleep state still consumes half the energy of the receive state.

The comparison of the figures above illustrates the impact of parameter values chosen for network simulations. In order to obtain an authentic sensor network model, one has to choose realistic and appropriate parameters and verify them with hardware. One has to incorporate implementation-specific issues, such as the longer transceiver switches with ScatterWeb, to obtain meaningful results that can be cross-compared to experimental results. When only considering the scarce information about the energy consumption of some node's components found on manufacturer datasheets, the simulation results will be restrained to an artificial horizon.

5.3.3 AODV Route Discovery Experiment

simulation parameters	
topology:	
area	300 m × 300 m
nodes	90 uniform distribution
routing parameters:	
routing header	80 *bit*
data packet payload	80 *bit*
RREQ retries	2
RREQ retry timer	3s
best instants k	2

We ported the k-best-instants broadcasting technique to the WiseMAC implementation in the network simulator (see 4.1), and integrated it with a popular on-demand routing protocol. The MAC layer broadcast mode is of high importance when implementing flooding mechanisms in on-demand routing protocols, such as AODV [14], DSR [15], but also in the data-centric Directed Diffusion protocol. Nodes aiming to transmit a packet will search their destination by initiating costly route request queries. Typically, the source node initiates such a cycle by emitting a route request, which is then rebroadcast in every intermediate node, until it either reaches the destination or an intermediate node knowing a path to it. We chose to use AODV as a well-established, efficient routing protocol, because the AODV one-hop paradigm fits well to WiseMAC with its schedule offset table of the one-hop neighbors. AODV is both energy- and memory-efficient. It neglects to transmit and store the full routing information between two endpoints. The route knowledge itself is distributed in the network, which makes sense in a resource-constrained wireless sensor network.

We tested out the performance of the upper schemes in an AODV route establishment scenario where every node in the network aims to find a route to the sink in a 90 nodes uniformly distributed topology on a 300 m × 300 m plane. In the following, the nodes first go through a bootstrapping neighborhood discovery process of a few seconds during which they find their respective neighbors by sending a few HELLO messages using the original WiseMAC full-preamble broadcast mechanism. After 1 minute, the first node emits a route request for the sink node, as it wants to start reporting data. The request is flooded over the network until reaching the sink, which answers the request. After receiving a route reply, the packet is forwarded hop by hop to the source by unicast. In intervals of 5 seconds later, one node after the other searches a path to the sink, until every node has found a route to the sink. The route request procedure is limited to 2 retry attempts, permitting up to 3 route request query cycles. After 500 seconds, the simulation is stopped, and the total energy consumption of all nodes summed up.

With both broadcasting techniques, every node managed to find a path to the sink and transmit the unicast packet. All route requests were successfully answered with a respective route reply.

As figure 5.18 illustrates, the k-best-instants approach already leads to an efficiency gain of $\sim 40\%$ for the overall energy consumption in this simple AODV route establishment scenario. The performance gain of the k-best-instants broadcasting technique weights as much as broadcasting and flooding mechanisms are used in the wireless sensor network. In an application scenario where queries are subsequently flooded to the nodes, using the

k-best-instants broadcast might also perform better than the original WiseMAC broadcast scheme.

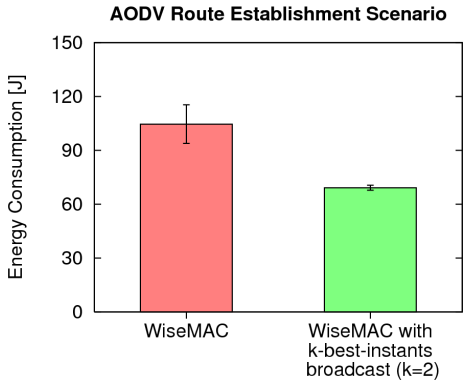


Figure 5.18: Performance of the broadcast schemes

The reason for the much better performance of the k-best-instants scheme is that with the full-preamble technique, every node in an ad-hoc network will have to listen to every other nodes rebroadcasting RREQ's. The extended carrier sensing range that is applied with WiseMAC also has a heavy impact. Nodes in this extended range will be kept awake with every preamble tone they hear. This leads to much more nodes staying awake than with the short transmissions of the k-best-instants technique.

5.4 Moving Intervals Wake-up Pattern

The WiseMAC protocol switches the transceiver between receive and sleep state in a simple periodic manner. In each cycle, the transceiver is switched to receive for a very short duty cycle that only serves to sense the carrier for the preamble signal. The time in-between two wake-ups however stays constant for each node. Once a node has been turned on, it starts alternating between receive and the sleep state with a constant period and individual wake-up pattern. When applying no synchronization measures at all, and nodes are not synchronously booted, the simple scheme leads to uniformly distributed medium samplings of the nodes over time. According to [5], *systematic overhearing*, as it occurs in synchronized MAC protocols like SMAC and TMAC, does only seldomly occur, as in most cases, the wake-up intervals of the nodes will not intersect.

WiseMAC designer *El Hoiydi* argues in [5] that non-synchronization inherently leads to a so-called *probabilistic overhearing avoidance*. Short transmissions between a sender and receiver pair will most probably not be heard by any other nodes, as all nodes have their independent sampling pattern. Short transmissions are likely to fall in between the independent sampling instants of potential overhearing stations.

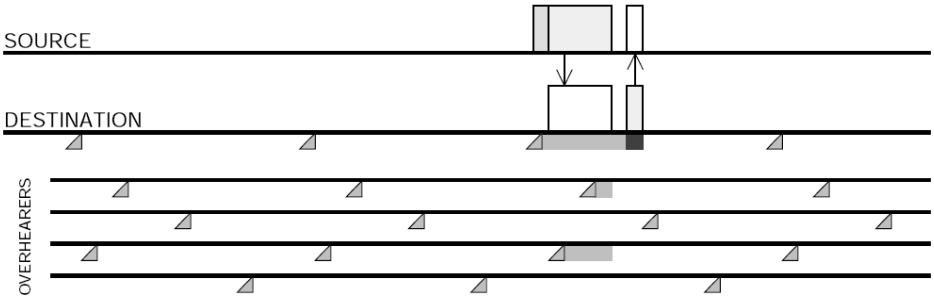


Figure 5.19: WiseMAC probabilistic overhearing avoidance

Figure 5.19 illustrates the property of the probabilistic overhearing avoidance. When a source sends a packet to a certain destination, the frame exchange is most likely not overheard by most of the nodes, as WiseMAC nodes sample the schedule in their independent wake-up pattern. Only two stations partially overhear the transmission in the figure, as they wake up and sample the medium during an ongoing transmission.

In this section, we show that the WiseMAC fixed periodic wake-up pattern can be disadvantageous and lead to severe problems when the preamble samplings of neighboring nodes are near each other. We especially show that indeed, *systematic overhearing* can occur with WiseMAC. We show that this problem can harm network service characteristics and we identify the root cause. We outline a mechanism and modification on the WiseMAC fixed periodic sampling pattern and prove that this scheme is suitable to deliver better results than original WiseMAC with increasing traffic rate. We underline our observations with experiments carried out on the simulator and on the sensor hardware testbed.

5.4.1 Drawbacks of the WiseMAC Static Wake-up Pattern

The fixed static wake-up pattern of WiseMAC leads to some drawbacks which are not apparent from a first point of view. The deployment of a wake-up and carrier sampling pattern with a constant period makes it *impossible* for nodes with non-intersecting wake intervals to learn about the presence of their local neighbors just by overhearing messages originated by them. As pointed out in section 2.1, systematic and repeated overhearing is a source of energy waste and should be avoided.

WiseMAC Rediscovery and Schedule Update Problem

When applying WiseMAC with its fixed static wake-up pattern, most of the nodes will have non-intersecting wake-ups. Short transmissions are likely to fall in between the independent sampling instants. The so-called probabilistic overhearing avoidance achieves that transmissions between other stations are in most cases not overheard, which is a positive issue. One drawback however comes with this property: to overcome the problem of discovering the local neighborhood, one will have to introduce a bootstrapping phase where neighboring nodes are discovered using some full-preamble WiseMAC broadcasts right after node deployment. The WiseMAC fixed wake-up pattern will not allow nodes to rediscover each other again by overhearing each others transmissions.
If the nodes want to retain knowledge about the presence, the current state and the activity of their neighbors, they periodically need to stay awake for the full cycle period T in order to rescan and discover neighboring nodes' transmissions, or request and distribute schedule updates with own full-preamble WiseMAC broadcasts. In addition, a single bootstrapping phase does not guarantee that all neighboring nodes are discovered. It is possible that the broadcast is interferred by other distant transmissions or fails due to bit errors. In some scenarios, networks have to deal with slight movement of the nodes, what can lead to nodes getting out of range and others getting into it. It is a conceptual advantage if the wake-up pattern allows periodic but infrequent overhearing, in order to maintain knowledge about the local neighborhood.

WiseMAC Near Wake Intervals Shadowing Problem

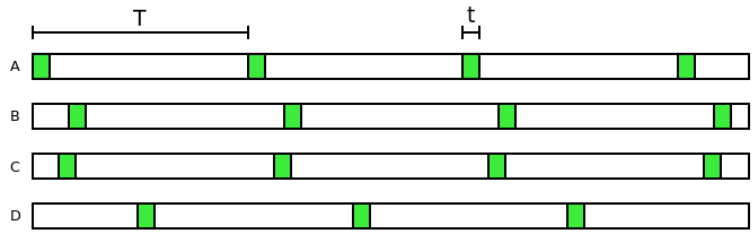


Figure 5.20: WiseMAC with cycle duration T and independent wake-up pattern

Another disadvantage of the WiseMAC fixed static wake-up pattern arises when there are several nodes with nearly identical wake-up pattern. In situations with increased traffic, such nodes systematically hinder each other from receiving messages. Consider node B and C in Figure 5.20, which share almost the same wake-up pattern. We assume that all nodes are at least in interference rage of each other. Node C always slightly precedes the wake-up period of node B. If two respective neighbors A, D want to reach B and C, the transmission $A \rightarrow B$ will always be shadowed by the transmission $D \rightarrow C$, as node C always wakes up earlier. D will always be capable of sending the preamble and start transmitting the frame to C, whereas B will wake up, notice that there is a transmission going on that is not destined to itself and go back to the sleep state after

the medium is idle again. A will have to wait until there is no message transfer to C such that it can finally transmit to B. This leads to a high latency for A's packets whenever there is traffic destined to C. We note that the wake pattern of C *shadows* the wake pattern of B and hinders it from receiving frames.

Notice that the later node B will *always* overhear messages that are destined to node C. When nodes transmit frames to C, node B will always wake up and overhear it. Depending on the preamble length and the gap in between the patterns of B and C, the danger that node C overhears messages or at least halve frames destined to B likewise is also present. The probabilistic overhearing avoidance of WiseMAC yields that neighboring nodes either *never* overhear each others transmissions or that they *constantly* overhear each others transmissions. This is an undesirable property, as nodes hearing every transmission of a neighbor will be prone to a much higher energy wastage and will drain out of energy first. Depending on their location in a wireless sensor network, this might have heavy consequences. The WiseMAC static wake-up pattern therefore inherently yields the danger of *systematic overhearing*.

The problem can have severe impact on the service properties for a large part of the nodes, especially if C and D are neuralgic spots in the WSN which have to forward data packets from whole subtrees. If D continuously generates or forwards packets, the traffic that needs to be forwarded by B is blocked. This can lead to high delays and even buffer overflows, and turns out to be a *fairness problem*. Furthermore, node B is likely to drain out of energy first, as it will have to stay awake for its own transmissions *and* the transmissions to C. The problem with WiseMAC is that the sampling pattern remains unchanged in every cycle. Once nodes shadow each other, they will shadow each other forever.

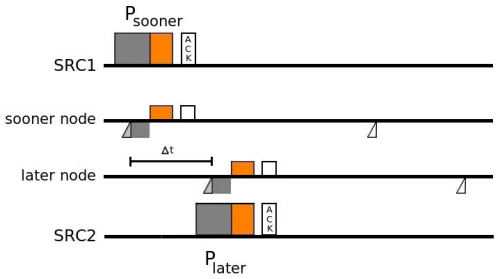


Figure 5.21: minimum gap in between wake intervals

The probability that wake patterns are too near depends on the wake cycle duration T, the bandwidth b and the size of the frames to be transmitted d. Figure 5.21 above depicts the case when a *sooner node* and a *later node* receive messages by their neighbors, SRC1 and SRC2. Their respective wake intervals yield exactly the desirable minimum gap $\triangle t$ such that the transmission to the *sooner node* does not hinder the transmission to the *later node*. As illustrated in the picture, the wake pattern of the *later node* yields a minimal gap of at least two halve preambles $\frac{P_{sooner}}{2}$, $\frac{P_{later}}{2}$, the duration of a frame transmission $\frac{d}{br}$, the turnaround delays of the transceiver t_{rxtx}, and the respective acknowledgement t_{ack}. If not, the transmission to the *later node* is not possible when there is a transmission going on to the *sooner node*. A sender would wake up, attempt to access the medium for transmission, notice that the medium is busy and back off and try in the next cycle.

Assuming that all node's wake intervals are uniformly distributed over time, we can

express the probability that the problem occurs that a *sooner node*'s wake-pattern shadows a *later node*'s wake pattern as the probability that two wake-patterns have a gap that is smaller than $\triangle t$. Given a random value t in the interval $[0, T]$, the probability that the value is smaller than $\triangle t$ is $\frac{\triangle t}{T}$. The probability that the shadowing problem occurs between two nodes therefore yields:

$$p = \frac{\triangle t}{T} = \frac{\frac{P_{sooner}}{2} + \frac{d}{br} + t_{rxtx} + t_{ack} + \frac{P_{later}}{2}}{T}$$

When assuming an inter-delay of 10 seconds and the parameters of the WiseMAC simulation of section 4.1, the probability that the *sooner nodes*' wake interval shadows the later node's wake interval is 6.92%. Likewise, the probability that the vice-versa case occurs is also 6.92%. The probability that one node shadows the other is therefore 13.84%, for any two neighboring nodes in the network.
Another drawback of WiseMAC's simple periodic wake pattern occurs when applying the k-best-instants broadcasting scheme described in section 5.3 to reactive RREQ-RREP query based protocols. When disseminating RREQ flood across a multi-hop topology, nodes will always consider the same nodes' intersections for rebroadcasting a frame. The network will stick to the same behavior in every retry attempt and will distribute the request always to the same set of nodes, and therefore probably not find appropriate routes. This is especially the case when there are bottlenecks in the network topology. Similar questions and topics are investigated furthermore in [3].

5.4.2 Moving Intervals Wake-up Pattern Concept

The WiseMAC fixed periodic wake-up mechanism that consists in sampling the medium with a common basic interval T can be improved in a quite simple manner. We can achieve a medium allocation scheme with similar properties but a better probabilistic overhearing avoidance by introducing a time movement function for the wake intervals. This function shall determine the instants for the wake intervals. An initially chosen cycle shall be kept as base for the movement function, but the medium samplings shall not always begin at the start of the interval.
There are different possibilities to integrate moving wake periods. We can introduce a moving interval in between two fixed intervals, similar to the mechanism proposed by *Braun et al.* in [2] with fixed and random wake intervals. The authors motivate the choice of a fixed wake-up interval repeating with a constant period and a random wake-up interval in between to solve the problem that nearby nodes are possibly never detected due to non-intersecting wake patterns. Our motivation for the choice of moving wake intervals is threefold:

- The behavior of a moving wake period is deterministic and follows a simple linear movement function which is identical and predictable in every node. If a node wants to transmit a frame to one of its neighbors, it determines its next wake interval by consulting its schedule offset table, where previously received wake-up announcements are stored and the next wake intervals of the respective neighbors are regularly calculated according to the movement function. The sender node can therefore determine the current position of the wake interval in the node its aims to deliver a message to. It prepends a preamble to the frame and then awaits its neighbor's wake-up. It appends its own wake-up announcement to the frame to update the receiver and likewise receives a schedule update from the receiver piggybacked on the acknowledgement. Everything is done exactly as in WiseMAC, the only difference is that the wake intervals follow a movement function.

- The problem referred to as *Rediscovery and Schedule Update Problem* in 5.4.1 that non-intersecting wake-up patterns could lead to nodes never discovering each other, is also resolved using moving wake intervals. Sooner or later, nodes will overhear frames or acknowledgements, even from or to nodes with non-intersecting wake-up patterns. The moving wake interval ensures that - given some periodic traffic - this will happen within a limited amount of time. In contrast to WiseMAC, nodes will infrequently overhear messages originated by their neighbors and will be able to adjust their neighbor schedule offset tables accordingly.

- The problem of nodes being *shadowed* by nodes with a slightly sooner wake-up interval and the *systematic overhearing* of each others transmissions described in 5.4.1 is solved in an elegant manner. By incorporating timely moving wake intervals, the situation that two nodes share almost the same behavior can still occur, but is far less probable than in the case with only a fixed wake-up interval. The situation that one node is shadwed by another nodes' wake-up pattern is less probable, as for such a case, the nodes would have to share the same wake-up pattern *and* the moving wake intervals would also have to be accurately in the same movement state in each timestep. Randomly choosing the initial configuration of the moving interval in every node makes this case less probable. The movement function of the moving wake period leads to a floating spreading of the node's wake periods over time, which is a more flexible and adaptive scheme than the fixed periodic wake pattern of WiseMAC.

In the following, we analyzed four variants for timely shifting movement functions, which are depicted in figure 5.22 and labeled I, II, III and IV. They differ in the function determining the instants of the wake-up intervals over time. All variants however work with the same average rate of wake-ups over time - the actual amount of wake-up intervals and time spent in receive mode is the same for all variants.
To simplify the illustration, a basic interval duration T is segmented into eight timeslots. Each timeslot has a duration t and represents the wake intervals to sample the medium. The four variants depicted below illustrate how the wake-up are assigned to the slots to accomplish the movement pattern.

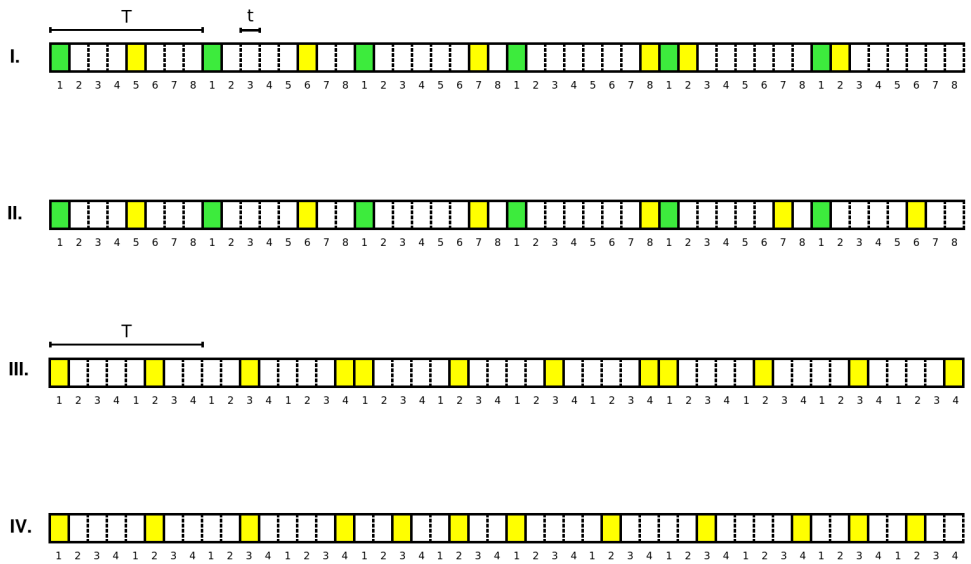


Figure 5.22: Variants: fixed wake periods (green) and moving wake periods (yellow)

- case I: *Fixed and Moving (forwards)*
  One possibility is retaining a fixed wake interval which starts constantly at position 1 in the fixed cycle, and integrating a moving wake interval in each cycle. The movement function is linear and only in forward direction. When the moving wake interval reaches position 8, it jumps back to position 2 and starts moving forwards again.
- case II: *Fixed and Moving (forwards and backwards)*
  The second variant we analyzed is closely related to the first. The only change consists in the movement function of the moving interval. The movement function in this case oscillates stepwise between states 2 and 8. When position 8 is reached, the interval shifts back to 7 and stepwise back to 2.
- case III: *only Moving (forwards)*
  The third variant consists in maintaining only the fixed cycle, but not using the beginning of the cycle for the wake-up intervals. It is the first variant *without* any fixed wake intervals. Movement is - as in the first variant - only in forward direction. In order to keep the service parameters comparable, the cycle duration is only half of the one applied in the first two cases such that the expected value of the wake-up frequency is the same in all scenarios and the wake-up patterns

remain comparable. In between T, there are still two moving intervals. To ease the illustration, we changed the numbering of the slot in figure 5.22. The intervals move forward and then jump back to the beginning of their cycle.

- case IV: *only Moving (forwards and backwards)*
  The fourth variant also has no fix wake-up interval, and differs from the third variant only in the movement function of the moving interval, which now moves forwards and then backwards inside the cycle.

To make our point clear, we deliver a combinatorial proof of the advantage of the moving wake intervals in the simplified model of figure figure 5.22. Let us assume that each node's cycle is segmented into slots to implement the movement function. When booting, every node starts alternating in its own wake pattern, but as a simplification, the start of every node's pattern exactly comes to be in one of the slots. This assumption is no limitation at all, it only serves to illustrate the mechanism that leads to a better *overhearing avoidance* with the approach of moving intervals, and helps to make our point clear. As slots can be of arbitrarily small duration, the assumption is not limiting the scope.

In figure 5.23, nodes follow a fixed static wake-up pattern as in WiseMAC. Nodes sample the medium in every fourth slot. The problem that two nodes share exactly or nearly the same sampling pattern, such that one node *shadows* the other, as described in 5.4.1, can be modelled in our simplified slotted model as the case when two nodes always sample the medium in exactly the same slots. This is the dangerous case we aim avoid, as the node's serviceability will become dependent from traffic to each other in this case, and nodes will be prone to *systematic overhearing* of each others transmissions and therefore spoil energy. Figure 5.23 depicts four possible combinations of two node's wake patterns. As we can see, the first case out of four is dangerous. The probability that nodes share the same behavior is therefore $\frac{1}{4} = 25\%$ in our model.

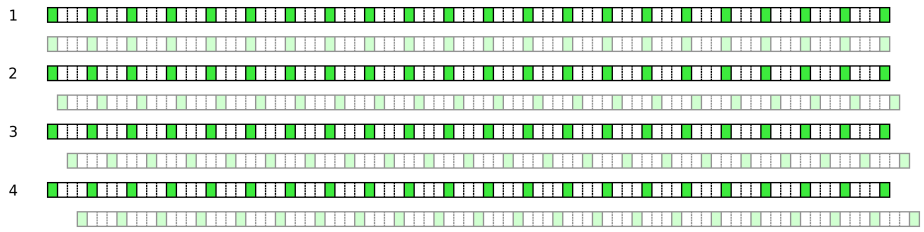


Figure 5.23: four combinations with the WiseMAC fixed static wake-up pattern

Figure 5.24 below displays the first 9 possible combinations of two nodes applying the wake pattern of *case I*. Nodes sample the medium with one fixed and one moving interval in each basic wake cycle T. The wake-sleep ratio is the same as in the WiseMAC case in figure 5.23, as every fourth slot is allocated for a wake-up in average. In contrast to WiseMAC, there are now a lot more combinations possible, as every nodes chooses the initial position of its initial fixed slot *and* of its moving interval randomly. There are 7×8 wake pattern combinations possible for two neighboring nodes applying one of them. The dangerous situation that two nodes share exactly the same behavior is only one of them. The probability that this case happens is only $\frac{1}{56} = 1.78\%$ in the slot model.

Figure 5.24: Fixed and Moving (forwards)

Thanks to the mobility of one wake interval, the probability that nodes share exactly the same behavior and constantly hinder each other from receiving messages, and *systematically overhear* each others transmissions is much lower. We have obtained a scheme that offers an overhearing avoidance that is more or less equal between arbitrary nodes, and have yet retained the deterministic nature of the wake intervals. The scheme does not avoid that there are wake-ups where two nodes are awake at the same time for some slots, but makes it improbable that nodes' wake-up patterns are identical such that they *constantly* wake up in the same slots and spend energy on overhearing each others traffic. In figure 5.24, one can observe that the count of slots that nodes share for the 9 first combinations averages to 2-3. It is better if there is an average count of shared slots rather than having either *no* or *all* slots in common. As mentioned, infrequent overhearing can be advantageous to correct relative clock drifts. We neglect to similarly analyze the three further cases II, III and IV in the simplified slot model, and instead test out their performance in the simulation environment in the following section.

5.4.3 Wake-up Pattern Simulation and Evaluation

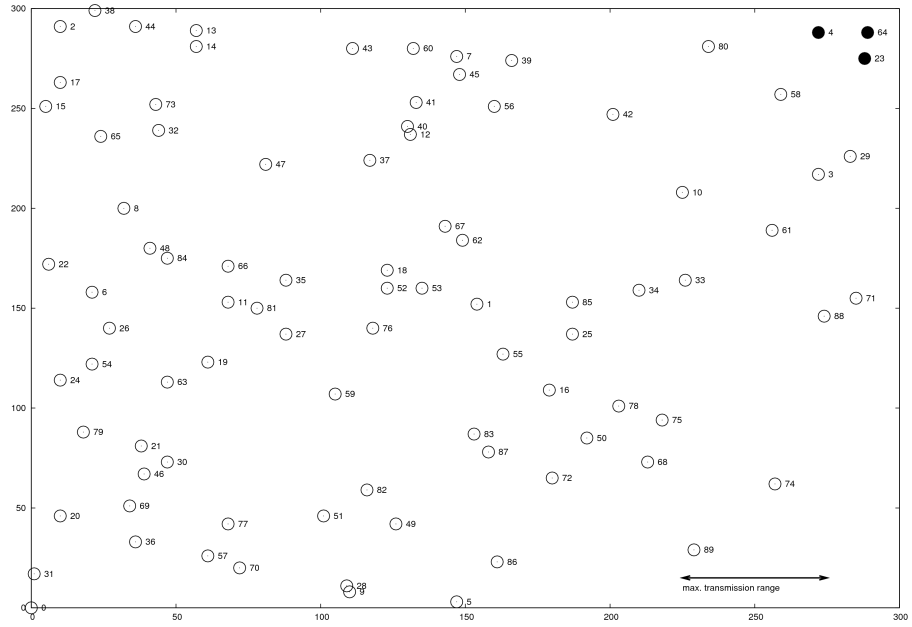


Figure 5.25: 90 nodes uniform distribution

In order to evaluate and compare the properties of the different cases I-IV suggested in the prior section, we deployed and simulated them on the 90 nodes uniform distributed network topology depicted in figure 5.25. We applied the parameters on the MAC layer outlined in sections 3.1 and 4.1. As the goal consists in the analysis of the properties of the four MAC-layer wake-up schemes alone, influences of a specific routing or broadcasting scheme were kept away by using static-shortest path routing. Applying a specific route discovery procedure could impose an undesired impact on the performance of the different movement schemes and tamper the results. By applying shortest path-routing, we inhibited these influences. Every node calculates the shortest path to the sink with Dijkstra's shortest path algorithm and picks its gateway node accordingly. If there is more than one gateway with the same cost, a node picks a gateway at random and keeps forwarding all packets over this node during the simulation run.

We compared the four approaches denoted as cases I, II, III and IV to the WiseMAC wake-up pattern with the fixed static wake-up pattern. In the case of WiseMAC, sampling interval is $T = 250\ ms$ between two wake-ups. For the four approaches I-IV we chose $T = 500\ ms$, but every node samples the medium twice in each cycle, as illustrated in figure 5.22. The expected value for the inter-delay then calculates as $\frac{T}{2} = 250\ ms$ and is therefore equal for WiseMAC and for all four variants I-IV. The only modification in comparison to original WiseMAC is the wake-up pattern, the arrangement of the wake-ups so to say.
After simulation start, nodes go through a bootstrapping phase of $20s$, where they all periodically emit a small amount of HELLO packets in order to discover their neighboring nodes. To make sure every node initially discovers its neighbors, this is achieved using full-preamble WiseMAC broadcasts. In this phase, nodes learn each others wake pattern and fill their neighbor schedule offset tables.

We investigated on the network characteristics in two simulation scenarios. In the first scenario, every node reports data to towards the sink station at position (0,0) in the lower left corner. By linearly increasing the traffic rate, we could observe how the transmission characteristics over multiple hops behaves with increased load.
In a second scenario, we analyzed the behavior of the network characteristics when triggering events on the topology. By linearly increasing the rate of events per time, essential insights could be gained.
The parameters of the simulation scenarios of the following sections are summarized in the table below.

simulation parameters	
area	300 m × 300 m
nodes	90 uniform distribution
routing	shortest path (dijkstra)
header	80 *bit*
payload	80 *bit*
simulation runs:	100
simulated time:	3600 sec

5.4.3.1 Node-to-Sink Periodic Traffic Scenario

In this scenario, we sticked to the most simple sensor network activity - the reporting of data with a certain rate of traffic by every node in the topology. Simple sensor networks i.e. for environmental monitoring purposes over a long time might have to fulfill such tasks. After the bootstrapping phase, every node starts sending packets over their shortest-path gateways. The gateways receive and forward the packets hopwise until they finally reach the sink. The sink station is energy-unconstrained, as it is assumed to be wired or fed with another power source.
Data traffic is generated by the node's respective application layers. The application generates traffic using the Poisson traffic model. The Poisson model is by far the most widely used and oldest traffic model in the literature to simulate network traffic. It is a simple and concise model that generates traffic of varying rate. Poisson traffic has a constant rate traffic over a long time and a bursty traffic behavior in short periods. In the Poisson traffic model, the parameter λ denotes the intensity of the traffic. With the Poisson traffic model, the expected value of packets during N seconds is $N\lambda$. We ran the simulation with constant reporting rate in each node and each run for values of $\lambda = 0.01, \ldots, 0.19$. All simulation runs last for 3600 s. After simulation time, we count together sent packets, received packets, dropped and lost packets.

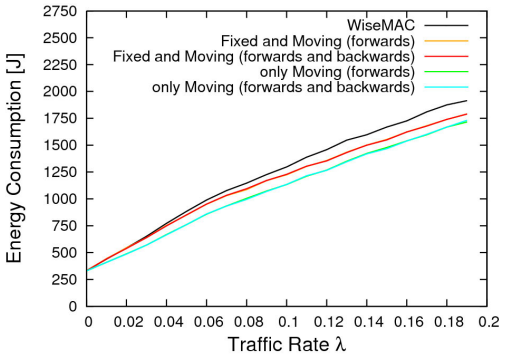


Figure 5.26: Energy Consumption

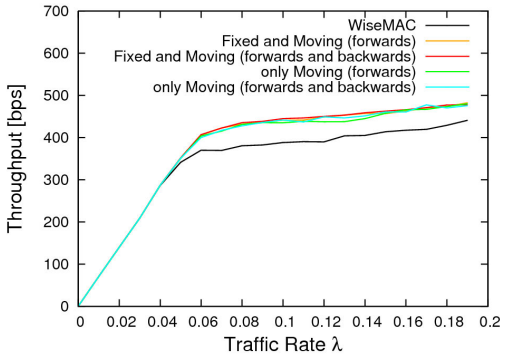


Figure 5.27: Throughput

Figure 5.26 depicts the sum of all energy consumed by all the nodes in the network together. It shows that all the approaches with moving wake intervals result in a slightly lower energy consumption than the approaches carried out with either one fixed and one moving interval or only fixed intervals in WiseMAC.
We investigated on the reason for this effect, which is however not easy to find, as the measured effect only sums up to approximately 5-10% difference for the whole network. We find that the problems arising with the fixed wake-up pattern account for the major difference. If there are wake-up patterns, the probability that there are nodes in the vicinity sharing the same or a similar behavior is much higher, as pointed out by the model proof in section 5.4.2. If this case occurs, then nodes will stay awake for every transmission destined to the node with the similar wake pattern and overhear it, which is costly. *Systematic overhearing* can indeed happen in WiseMAC, and is not that improbable as it may seem from a first point of view.
The probability that two nodes share the same behavior and tamper each other is raised in WiseMAC by the application of the extended carrier sensing range. Interference from many stations inside this range can lead to nodes being kept awake. Having only moving intervals, the probability that the stations share the same behavior is much lower. What might not be well-visible as the curves are almost not distinguishable from each other is that both mechanisms with one fixed and one moving (I,II), as well as both approaches with only moving intervals (III, IV) perform almost identically. Obviously, the two approaches with only moving intervals perform best in respect to the overall energy consumption, as clearly visible in figure 5.26.

Figure 5.27 illustrates the throughput with increasing traffic rate. Comparing the approaches I-IV with the original WiseMAC wake-up pattern, we can claim a slight performance improvement with increasing traffic rate, which however remains in the range of 10-15%. It is insightful that the mechanisms incorporating moving wake periods are only measurable with increasing traffic. As long as there is not much traffic, the situation that two stations need to be contacted with a similar wake-pattern does not yet occur. With low traffic, stations can wait for one cycle if one is shadowed, and then transmit their packets. This effect is yet already measurable in the energy discussion in figure 5.26. With increasing traffic, the problem that two nodes with similar wake-pattern need to be contacted concurrently, as they might need to forward packets from their subtrees, leads to congestion problems. As illustrated in 5.27 congestion problems arise earliest with the fixed static wake-up pattern of WiseMAC. In respect to throughput, the approaches I, II, III and IV do not differ at all. The only conclusion one can draw is that the introduction of moving intervals has led to a slightly better throughput. The better overhearing avoidance impacts similarly on throughput and energy consumption.

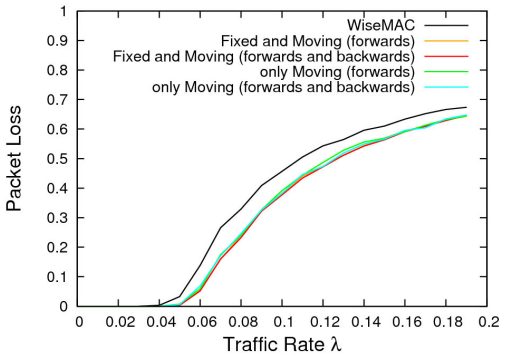


Figure 5.28: Packet Loss

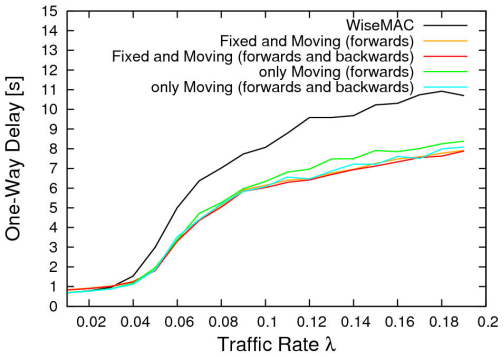


Figure 5.29: One-Way Delay

Figure 5.28 and 5.29 depict packet loss and the average end-to-end latency of the packets when arriving at the sink station. The figures confirm our conclusions and observations. The fixed static wake-up pattern of WiseMAC again performs worse than the approaches that incorporate at least one moving interval. Again, the better overhearing avoidance between any two neighboring nodes is responsible for the efficiency and performance gains. In respect to the one-way delay, the performance gain comes to 20-30% with increasing traffic rate λ. This is a direct consequence from the congestion problems that arise earlier in WiseMAC. If there are crucial nodes that need to forward a lot of packets near the sink, it comes to congestion problems when they share a similar behavior. In respect to packet loss, some improvement of 5-10% could be claimed.
It seems that retaining a fixed wake interval is obsolete and the solutions without any static wake-up interval should be favored. When keeping a fixed wake interval, the situation where one node is shadowed by another can still occur, although it is much less probable than with WiseMAC. The overhearing avoidance of the approaches with only one moving interval seem to perform best. Cases III and IV with only moving intervals shifting in their cycles yield the best performance in respect to overall energy consumption and one-way packet delay.

5.4.3.2 Node-to-Sink Distributed Events Scenario

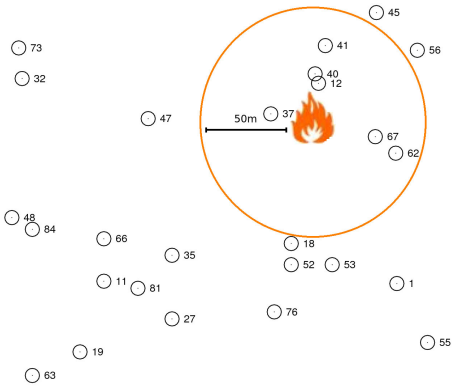


Figure 5.30: Event on the 300 $m \times 300$ m plane

Similar results as in the periodic reporting scenario of section 5.4.3 could be observed in a distributed events scenario. On the same network topology, we triggered *events* on randomly chosen spots on the 300 $m \times 300$ m plane. When an event happens in such a position, each node in the vicinity of 50 m starts reporting data with between 1 and 3 packets. In figure 5.30, an extraction of the topology is illustrated. A fire indicates the event position. In this case, nodes 37, 12, 40, 41, 67 and 62 would start reporting data packets to the sink. Again, we applied Dijkstra's static shortest-path routing algorithm to omit influences from the routing scheme. Using this scenario setup, we aim to model applications where sensor networks are used for event-detection and monitoring purposes, i.e. fire detection, movement detection or object tracking.
In event-based sensor networks, events trigger data to be reported from nodes in the actual vicinity. There are techniques to aggregate and preprocess data inside the network and only deliver results to the sink. In this simplified scenario however we omitted to do so. Every packet is hopwise sent to the sink node.

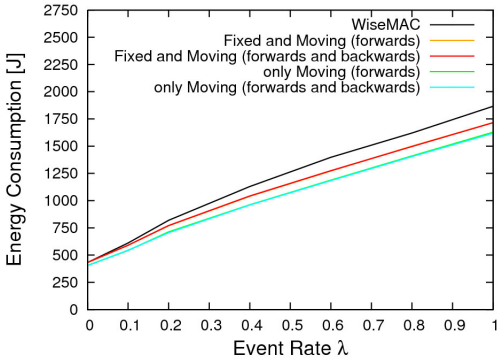


Figure 5.31: Energy Consumption

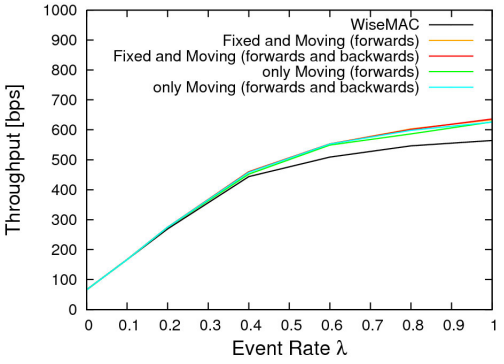


Figure 5.32: Throughput

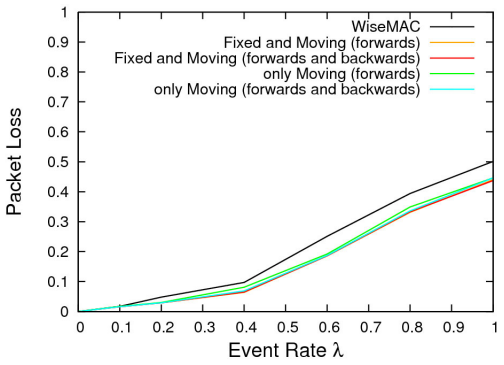


Figure 5.33: Packet Drops and Losses

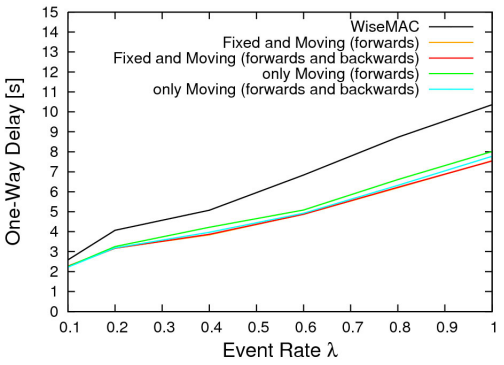


Figure 5.34: One-Way Delay

The results overall approve the observations and conclusions of the periodic traffic scenario in the prior section (5.4.3). We can observe the best operation characteristics when applying the mechanism with only one moving interval (Case III and IV in 5.4.2). These approaches perform better than WiseMAC in regard of throughput, delay and packet loss, and are energetically slightly superior to the approaches with one fixed and one moving interval (I,II).
We conclude that the integration of moving intervals has a slight but nevertheless positive impact on the network characteristics in respect to energy-consumption, throughput, latency and packet loss. In the next scenario, we aim to prove the simulated efficiency gain in a simplified scenario and an implementation on real sensor hardware.

5.4.4 Moving Intervals on the Embedded Sensor Boards

We implemented the scheme with one moving interval moving inside a tact window and referred to as *case IV* in 5.4.2 on the Embedded Sensor Boards platform. The wake interval is now segmented into 128 slots. Depending on the basic wake interval duration, nodes shift their wake-ups forward and backward in respect to the beginning of each cycle for a certain timedifference $\triangle t$ which calculates as $T/128 = 3.9\ ms$ or exactly 4 ticks of the DCO internal clock (1024 ticks = 1 second). However, the slot structure only serves to accomplish a precise and transmissionable movement pattern, nodes still begin alternating in their own wake-up pattern and the cycle and slot starts of all individual nodes are not synchronized at all. In order to *tell* other nodes how and when their next wake-up will take place, nodes now transmit an additional byte in each packet, signalling the current state and the direction of the inter-temporal movement-function to their neighbors. The byte's semantics is organized as outlined in figure 5.35.

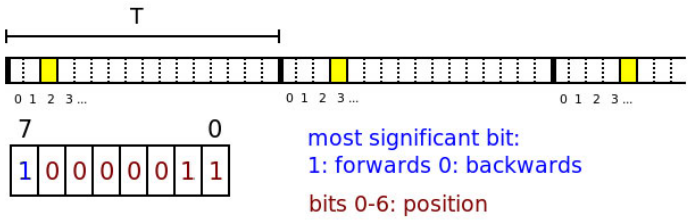


Figure 5.35: Basic wake cycle interval T segmented into slots

The most significant bit denotes the movement direction of the wake intervals. If the node shifts forwards in the next timestep, it sets the bit to '1', or '0' if it shifts backwards. The bits 0 to 6 denote the slot the node will allocate for its wake-up in the next cycle. The node of figure 5.35 is currently in the first cycle in the slot indexed 2, and the respective byte is displayed. If during the first cycle the node transmits a packet to a neighboring node, it appends the beginning of the next cycle *and* this byte, and thereby announces that its next wake-up will take place in the slot with index 3, and that it is moving *forwards*. The neighboring node can now determine the state of this node's wake interval in the future.

```
> ptb
> printing WiseMAC neighbor table
>
> neighbor table:
>
> ID      |LUPS    |LUPM    |NEXS    |NEXM    |STATE   |FWBW    |X       |Y
> 23      |429     |704     |446     |137     |7       |1       |0       |0
> 17      |423     |125     |446     |382     |98      |0       |0       |0
> 4       |421     |824     |446     |293     |27      |0       |0       |0
>
> current time: 446 secs - 122 millis
> [ptb] OK
```

Listing 5.1: print table command output

Listing 5.1 depicts a neighbor table of a node as printed out by the print-table command (ptb) on the ESB prototype implementation. Nodes regularly update their neighbor table and estimation about the start of the next cycle of their neighboring nodes. Columns NEXS (=Next Tact Seconds) and NEXM (=Next Tact Millis) contain the next-soonest beginning of the neighboring nodes' cycle. The listing lists three neighbor entries with ID's 23, 17 and 4. The column STATE contains the information about which slot is being allocated in the next cycle. The FWBW column determines the direction of the inter-temporal movement function, '1' for forwards and '0' for backwards.

5.4.4.1 Near Wake Intervals Experiment

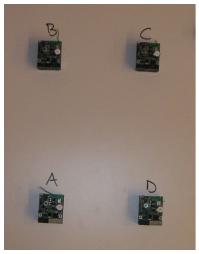
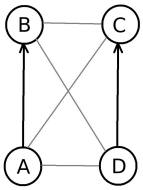


Figure 5.36: scenario setup



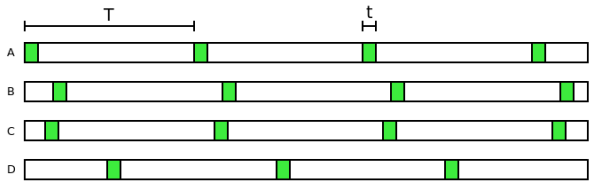


Figure 5.37: WiseMAC nodes with near wake intervals

The WiseMAC fixed static wake-up pattern can be disadvantageous when there are nodes with near wake intervals and similar sampling pattern in range of each others. The problem was referred to as *Near Nodes Shadowing Problem* and is described in 5.4.1. In this section, we aim to prove that the solution with the moving wake intervals solves this problem and provides a better and more reliable overhearing avoidance. We show that *systematic overhearing* of neighboring node's transmissions can be avoided, in practice, with the ESB implementation of the moving intervals scheme.
The setup of the experiment is outlined in 5.36. All nodes A,B,C,D are in range of each other. Technically, they build up a full mesh, as indicated by the grey links, but only the links in bold black are used. In a multi-hop topology with many nodes, the situation is quite probable to happen, as it is not necessary that all nodes are in range of each other that the problem occurs. It suffices when nodes A is in interference range of C and D in interference range of B. When node C's wake interval is slightly before another node B's interval, as depicted in 5.37, transmissions to C will always shadow and tamper transmissions to the later node B. Node B will always wake up and overhear transmissions destined to node C and spoil energy on *systematic overhearing*.

The probability that this problem arises is determined by environmental parameters and is analytically discussed in 5.4.1. We ran 100 tests by independently booting two nodes and testing if their respective wake pattern hinders each other from receiving messages. We found that the probability that this problem happens on the ESB is *significant*. Out of 100 independent tests, the problem occurred 23 times. As discussed in 5.4.1, the problem is approximately as probable as the duration of a entire packet transmission including the acknowledgement in relation to the duration of the basic wake cycle T. When applying again the *comparison* parameters, the duration to transmit one packet *and* respective acknowledgement denoted as $t_{transmission}$ calculates as

$$t_{transmission} = E(t_{MRP}) + t_{MP} + t_{frame} + t_{rxtx} + t_{ack}$$

$$\sim 3\ ms + 5\ ms + 20\ ms + 4\ ms + 10\ ms = 42\ ms$$

Where t_{MRP} denotes the medium reservation preamble, t_{MP} the minimal preamble and t_{rxtx} the transceiver switch from receive to send. The probability that the first node hinders the second therefore yields as $\frac{42\ ms}{500\ ms} = 8.4\%$. Again, we have to double this value as the vice-versa case might happen also. The analytical probability therefore comes to 16.8%, which is again a value that is not negligible, as the problem may occur between any two nodes in the network with this probability. The reason why we measured a value of 23% might be because in reality, there is a bigger gap in between frame and acknowledgement transmission than indicated with the analysis above. On the Embedded Sensor Boards WiseMAC prototype, a frame transmission with 12 bytes payload and the

respective acknowledgement comes to approximately $50 - 60\ ms$.

In this simplified scenario, we show that the static wake-pattern of WiseMAC leads to bad service characteristics for nodes of type C with increasing traffic rate. We show that the problem can be solved or at least softened with the moving intervals approach with only one timely moving wake interval moving forwards and backwards, denoted as case IV in the prior sections. We focus our investigation solely on the case *if* the situation of near intervals occurs. By synchronizing the nodes B and C in every experiment run and letting them choose a slightly different wake-up pattern, we achieved that the wake interval of C always precedes the respective wake interval of B by some milliseconds. We then generated traffic from nodes A to B and node D to C and measured the service characteristics of the traffic arriving at the *sooner node* C and the *later node* B. Traffic is generated according to a Poisson process of rate λ. We carried out the experiment with different values of λ ranging from 0.1 to 0.8 packets per second.

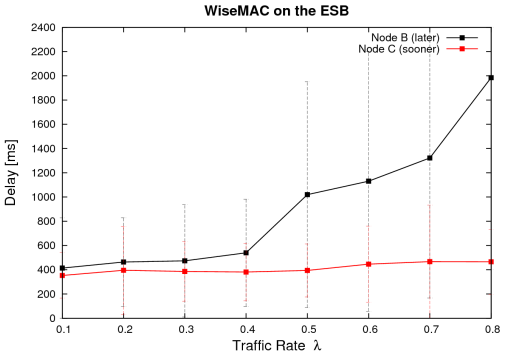


Figure 5.38: WiseMAC static wake-up pattern on ESB nodes

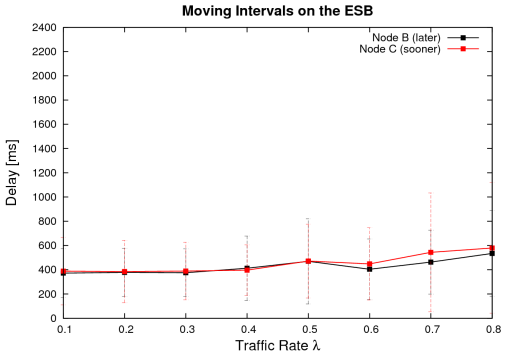


Figure 5.39: Moving Wake Intervals on ESB nodes

Figures 5.38 and 5.38 depict the behavior of the end-to-end delay of packets generated by the application layer in node A and D arriving at nodes B and C. Again, the delay is measured as the time the application generates the packet and the time the receiver node application layer receives end decapsules the packet.
The mechanism that now develops is the following: in any case D transmits a packet to C, node A can not deliver own packets to B and has to backoff and wait for the next cycle where D has no traffic to send. Node B's received packet delay therefore increases steeply with increasing traffic rate, as traffic to it is shadowed by node C's wake-up. As expected, C does not suffer from increasing delay with increasing rate λ, as its wake-up precedes the one of D. At $\lambda = 0.8$, we stopped the experiments, as the generated packets led to buffer overflows in A. Lacking of memory, the buffer only suffices to store up to 5 packets with the MAC header and the small payload of 12 bytes.
As displayed in 5.38, the MAC scheme with the Moving Intervals indeed performs better. As the wake interval shifts inside a cycle in respect to the movement function, chances are low that the two receiving nodes B and C *systematically* hinder each other from receiving messages. The latency of packets destined to both nodes is not distinguishable. The moving scheme leads to a more robust probabilistic overhearing avoidance. In contrast to the fixed static pattern of WiseMAC, it is less probable that both the start of the cycle *and* the initial configuration of the nodes' wake intervals lead to the problem of permanently near wake intervals. It leads to the property that the wake intervals come near each other and cross each other infrequently, which however is not so bad as nodes may exploit these moments and overhear each others wake schedules.

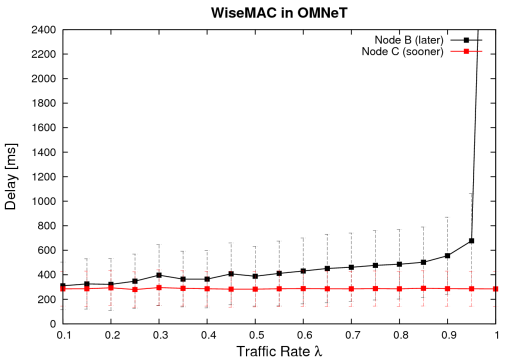


Figure 5.40: WiseMAC static wake-up pattern in OMNeT

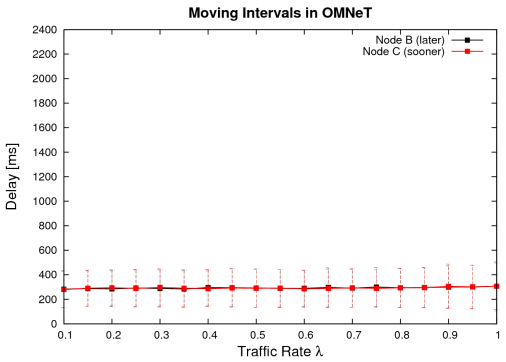


Figure 5.41: Moving Wake Intervals in OMNeT

Figures 5.40 and 5.41 depict the results of the same experiment setup carried out in the OMNeT simulator when applying the *comparison* parameters. Again, the wake interval of node C slightly precedes the respective interval of node B. The negative impact for node B being slightly later than C is clearly visible in the graph of node B's delay. The latency increases more or less linearly with increasing traffic rate λ until it steeply increases in the value of 0.95. This being a bit later than in our measurements on the ESB testbed. However, simple scenario shows in 5.41 that the scheme can be improved with the Moving Intervals approach. As in the measurements on the ESB nodes, transmissions to both nodes then have a comparable latency.
The Moving Intervals scheme allows that two nodes sooner or later wake-up in the same slot. This effect is clearly visible with the ESB nodes, as it happens every 128 slots or $128 \times 500\ ms = 64\ secs$, when their movement function reaches the end of the cycle and changes direction. When generating some traffic from A to B and D to C, this effect led to nodes discovering each other after some time by overhearing messages and their respective acknowledgement during the intersection. The nodes therefore aggregate knowledge without any further cost, just by exploiting the infrequently occurring overhearing.

5.4.4.2 6 Nodes Chain Experiment

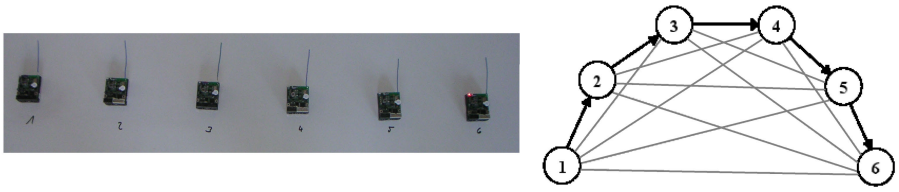


Figure 5.42: Chain topology

In order to analyze the behavior of the Moving Wake Intervals scheme with multi-hop traffic, we again chose the scenario setup with the 6 hop chain of section 4.3. We analyze the performance of the Embedded Sensor Boards Moving Intervals prototype and cross-compare the results with corresponding results of measurements with the same experiment in the OMNeT simulation and the corresponding results obtained with the WiseMAC prototype in section 4.3. Again, nodes are all in range of each other, and technically build up a full mesh topology, but only the links that are painted in bold

solid black are used.
We measured the one-way delay of packets being generated by the application layer in node 1 and being hopwise transmitted towards node 6. Traffic is generated with rate r and a jittering scheme is introduced to omit that results are effects of a certain periodicity of the traffic pattern. The inter-delay $\triangle t$ between two packets calculates as:

$$\triangle t = \frac{1}{r} + d_{jitter} \qquad d_{jitter} \sim uniform[+T, -T]$$

Figure 5.43 again depicts the results of the same experiment run with WiseMAC in section 4.3, and figure 5.44 the results of the Moving Intervals approach. Both figures illustrate the corresponding delays obtained on the simulator and on the ESB implementation. Again, we applied the *comparison* model parameters. Obviously, simulation and measurements on the ESB prototype are not that far from each other. In each figure, the simulator and ESB curves' differences are in the range of only around 5% or even lower. The reason for the difference between modelled experiment and the experiment on the ESB might be that a transmission on the ESB is still a bit longer as the *comparison* parameters actually model it in OMNeT, due to the delay in between frame transmission and acknowledgement. Some implementation-specific issues may also play a role.

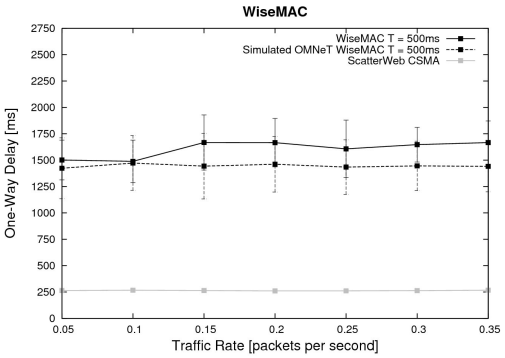


Figure 5.43: One-Way delay with WiseMAC static wake pattern

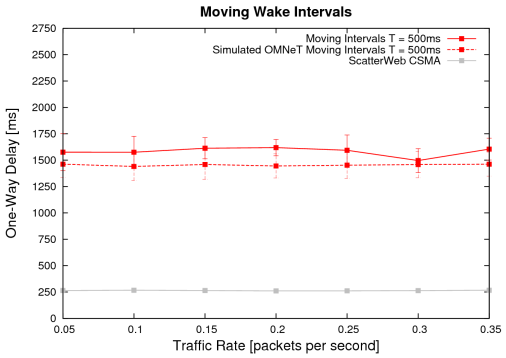


Figure 5.44: One-Way delay with Moving Intervals

Figures 5.43 and 5.44 both illustrate a delay of $1500 - 1600\ ms$ for the transmission over 6 hops, or a 300 ms per-hop delay. When moving from the WiseMAC sampling pattern to the Moving Intervals, one can not expect much to change in this delay. As the wake intervals move in between the same cycle of duration $T = 500\ ms$, the end-to-end latency still depends on the transmission delay and on the time a node has to wait for the next wake-up of the next node in line. With wake intervals moving inside a fixed cycle, the expected time to wait for the next instant therefore remains $E(t_{wait}) = \frac{T}{2} = 250\ ms$.
Again, we estimate the expected delay per hop $E(d_{hop})$ as the time necessary to wait for the next wake interval of the receiver node plus all delays that are necessary for frame transmission and acknowledgement, i.e. the time for the medium reservation preamble t_{MRP}, the minimum preamble t_{MP}, the transmission of the frame t_{frame}, the transceiver switches t_{rxtx} and t_{txrx} and acknowledgement t_{ack}. As we again applied the *comparison* parameters of section 3.3, we can incorporate the expected value $E(t_{MRP}) = 3\ ms$ of the medium reservation preamble t_{MRP}, which is a uniformly distributed value in between [0,6]. When incorporating all particular delays of the transceiver switches of section 3.3 into the calculation, we analytically obtain an average per-hop delay of

$$E(d_{hop}) = E(t_{wait}) + E(t_{MRP}) + t_{MP} + t_{frame} + t_{rxtx} + t_{txrx} + t_{ack}$$

$$\sim 250\ ms + 3\ ms + 5\ ms + 20\ ms + 4\ ms + 2\ ms + 10\ ms = 294\ ms$$

As the analytically obtained value does not change when applying the Moving Intervals approach, we do not expect the mechanism to deliver different results than WiseMAC. This expectation is met when comparing figures 5.43 and 5.44. The Moving Interval leads to more or less the same results as the original WiseMAC scheme in respect to the average one-way latency.

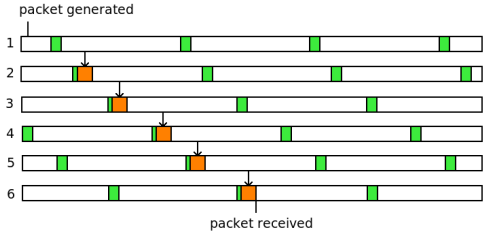


Figure 5.45: Advantageous WiseMAC wake-up patterns

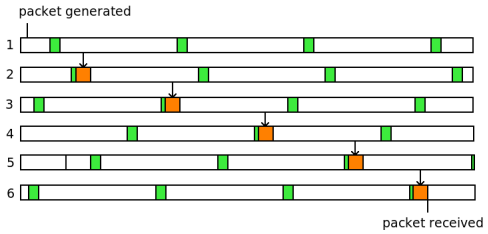


Figure 5.46: Disadvantageous WiseMAC wake-up patterns

Yet another effect could be observed in both simulation and measurement results, and is clearly visible in figures 5.43 and 5.44: The variance among the simulation run values for the one-way delay is smaller with the Moving Intervals approach. The significance intervals in both curves are approximately twice as long with the WiseMAC approach than with the Moving Intervals approach. This property can be explained as follows: Figures 5.43 and 5.44 depict the results of 20 independent experiment runs for each traffic rate. Independent because in each run, all nodes were rebooted and every node began alternating between sleep and wake states in its very own independent wake pattern.
With WiseMAC, the one-way delay is very much bound to the wake-up pattern of the intermediate nodes. In one run, the pattern of the intermediate nodes between node 1 and node 6 favors the transmission in the direction $1 \rightarrow 6$, as illustrated in figure 5.45. Every node only has to wait a few milliseconds until the next node's next wake-up, which results in a quite short one-way delay for this simulation run. With WiseMAC, this pattern stays fixed in the whole experiment run. The delay of a experiment run is therefore determined to a large degree by the wake-up pattern of the intermediate nodes.
In the next simulation run, the intermediate nodes' wake-up pattern might not favor the transmission along the chain, as the wake-ups might be quite far from each other. Figure 5.46 illustrates such a situation. The one-way delay is in this case much higher. Obviously, when applying WiseMAC, the different wake-up patterns in independent experiment runs therefore result in very strongly differing end-to-end latencies for the single experiment runs, which results in a stable average, but a high variance.

When running the 20 experiment runs with Moving Intervals, the wake-ups follow the movement function inside a fixed cycle. The inter-delay between two nodes wake-ups therefore changes between each packet delivery along the chain, as the wake-up intervals move inside a cycle. For one packet, the state of the intervals might just favor the transmission along the chain, whereas for the next packet, it might take a bit longer. Overall, the Moving Intervals approach leads to a more balanced one-way delay among the experiment runs, which results in lower variance in the observed values. This effect can be observed when comparing the standard deviations of the observations with the WiseMAC approach in Figure 5.43 with the standard deviations of the observations obtained with the Moving Intervals approach in Figure 5.44. The effect was encountered both in simulations and with the ESB prototype.

We found that measuring the lifetime of the nodes applying the Moving Intervals approach on the ESB is obsolete, as the only difference to the WiseMAC implementation that somehow requires more energy consists in the single byte being transmitted to keep track of the state of the moving wake interval (see figure 5.35), and the update mechanism of the neighbor table. The additional cost for transmitting one single byte and increasing and decreasing the state variables of some neighbor entries is negligible, and certainly not measurable at all with the methodology applied in this thesis (section 3.2.4). The superiority of the Moving Intervals scheme in respect to throughput and latency observed in the simulator would anyway not be observable in this small scenario. One would need more nodes and higher traffic rates, as the effect only pays off when signs of congestion arise.

5.5 Traffic Adaptivity

WiseMAC's approach of lowest-possible duty cycles and the preamble sampling technique proved to be a very efficient communication scheme. As WiseMAC applies no channel reservation to mitigate the hidden node effect, and solves this problem with the cheaper carrier access policy of the so-called extended carrier sensing range, no additional control messages are necessary. The only pure control message in WiseMAC is the acknowledgement frame. Overall, WiseMAC proved to be suitable and more energy-efficient for application in wireless sensor networks than existing WSN MAC protocols under variable traffic conditions, such as S-MAC or T-MAC. Many sensor MAC proposals that have occurred since the suggestion of WiseMAC have adapted and integrated variants of the preamble sampling technique, also called low-power-listening.

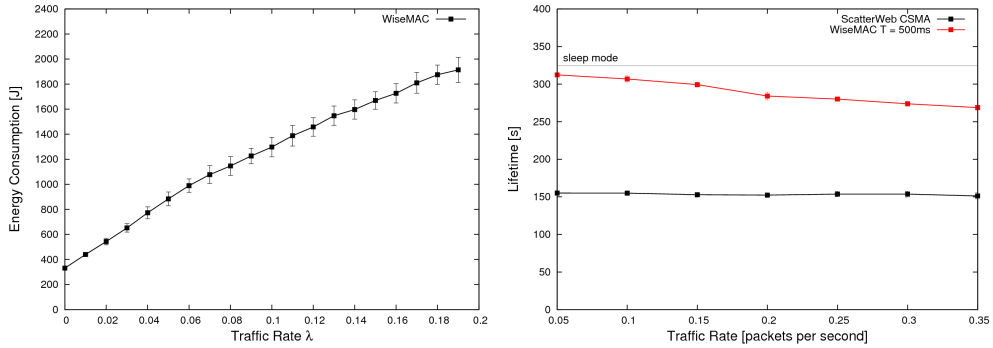


Figure 5.47: Energy Consumption with WiseMAC

WiseMAC achieves a main goal of energy-efficient MAC protocols - it manages to use the radio interface in an *on-demand* manner. If no traffic has to be sent, the protocol overhead is very low. It consists in the simple periodic preamble samplings. WiseMAC renounces on costly centralized or distributed synchronization schemes. More energy is only spent when actually traffic is being handled. The good traffic adaptivity of WiseMAC is clearly visible in figure 5.47, which depicts the overall energy consumption with the original WiseMAC approach carried out in the experiment of section 5.4.3 on the left and the lifetime results of the WiseMAC ESB prototype on the right. With no traffic, the energy consumption remains very low. With linear increase of traffic, WiseMAC is able to react with a more or less linear increase of the total energy consumption, which is a desireable property.

Problems arise when dealing with packet bursts and when neighboring stations are also intending to send traffic. One problem that can arise is quite similar to the early-sleeping-problem occurring in TMAC (as described in 2.1.2). When a node wants to reach a station in its wake interval, but fails to gain access to the medium, may it be because there is another station sending to the node or another distant sender is currently blocking the medium, it is quite probable that the preamble sampling period is missed and that the destination node goes back to sleep *too early*. The very short duration of the duty cycles to sense the carrier has a vast impact on the maximum traffic rate. By limiting the duration of the sleep intervals to only a few percent of the cycle interval, the boundary values for the maximum traffic rate are determined. If there are many senders aiming to transmit packets to one receiver, the transmission rate is more or less limited to one packet exchange per wake-up. WiseMAC only offers a simple concept to signalize pending transmissions between one sender and one receiver, the so-called *more bit*.

5.5.1 Increasing the Duty Cycle

Aiming to make the WiseMAC protocol more adaptive for varying or increasing traffic load, we followed the idea to let nodes react with increasing duty cycles when the traffic load exceeds a certain threshold. We let nodes react to the increased load by doubling or tripling their duty cycles. It soon turned out that an increase of the duty cycles alone does not help at all, but only causes additional costs. The problem is the intrinsic information asymmetry of the communication problem. Nodes increasing their duty cycles will have to actively notify this increase to their neighbors.
We simulated a stepwise increase of the duty cycles when the incoming packets during the last few cycles exceeded a certain threshold. But as every node assumes that its neighboring nodes sample the medium with the minimum default sampling period, a node increasing its duty cycle has to notify the increase to its neighbors, and transmit a message for *every* neighboring node, as every neighbor follows its *own* wake-up pattern - or prepend a preamble of the duration of the main interval T to reach all neighbors. We tried to broadcast the increase of the duty cycle with the broadcast concept of the best instants. The approach turned out not to lead to an improved traffic adaptivity and energy efficiency, but to massive control messages being generated right when it was least desired. More energy was spent for control messages and overhearing. The service characteristics delay and throughput could not be improved, on the contrary the mechanisms performed worse than the WiseMAC scheme. We ceased the investigations on this idea and pursued another approach.

5.5.2 More Bit and Extended More Bit

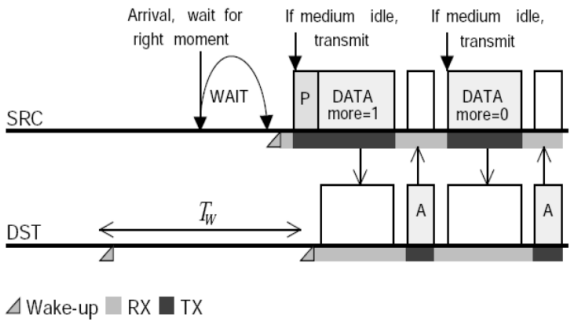


Figure 5.48: more bit scheme in WiseMAC

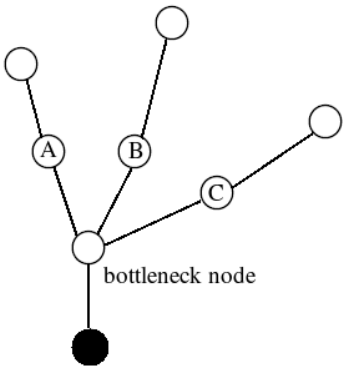


Figure 5.49: Tree structure in a wireless sensor network

To increase the maximum achievable throughput in case of packet bursts and higher traffic load, WiseMAC suggests an optional fragmentation scheme called more bit mode. WiseMAC sets a flag bit in a unicast MAC frame whenever a node has *more* packets to send. The more bit in the frame header signalizes to the receiving node that it shall not turn the transceiver off after the reception and acknowledgement of the frame, but switch to the receive mode again in order to receive the next packet. A sender thereby does not need to wait for the next wake-up of the receiver to transmit the next frame, what leads to a throughput increase. The scheme proved to be very effective in scenarios with varying traffic, especially with packet bursts generated by single nodes.

We tried to adapt the more bit idea one step further. The scheme only serves to improve traffic adaptivity between one sender and one destination, and therefore has its limitations. In a WSN scenario, there are often nodes which have to forward data from large subtrees. Such bottleneck nodes have to forward messages generated by many other nodes, supposedly large subtrees in the routing hierarchy, such as the node depicted in 5.49. The more bit scheme does not help at all if more than one node aim to concurrently transmit a packet to a node. One after the other will have to wait for a wake-up of the bottleneck node in order to forward their frame. We aimed to develop a strategy where nodes will automatically stay awake for a longer time than just the sampling period, when more traffic has to be handled, and tell and signalize this to all nodes waiting to forward traffic to it, such that the information asymmetry is resolved. We therefore extended the semantics of the more bit to an *extended more bit*, accomplished by the *stay awake promise* of the receiver.

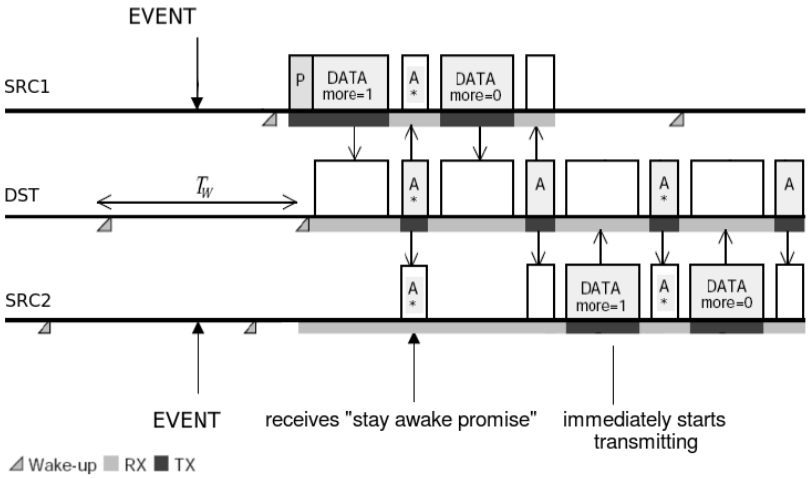


Figure 5.50: Extended more bit scheme based on stay awake promise

The scheme is depicted in Figure 5.50. The illustration shows two sources SRC1 and SRC2 simultaneously aiming to transmit some packets to the same node DST, possibly because an event has occurred. If SRC1 and SRC2 both aim to reach DST in the same wake interval, the medium reservation preamble will decide who is first. SRC1 wins the contention and sends its first two frames with the more bit set. The destination node acknowledges the more bit in the ACK packet and stays awake for at least one basic wake interval T. As SRC2 has lost the contention, it will wait and overhear the transmission to DST. By hearing the stay awake promise in the ACK, SRC2 knows that it can start sending its own data frames right after SRC1 has finished its transmissions, as it knows that DST will stay awake.
The advantage of this scheme is that no time of the DST node is wasted, as the transmission of SRC1 can start immediately after node SRC1's transmission. The mechanisms however only gets in charge when there is a node buffering more than one frame, which is a signal of increased load. The scheme is certainly not applied after every unicast transmission, as this would cause much energy waste and rather not be energy-efficient, especially not when the traffic is of only low rate.

The results of this section are published in [12]. The scheme might be further investigated to account for other signs of increased load or congestion, such as failing transmissions due to interferences, the overhearing of retransmission attempts or other detected overloading situations.

5.5.2.1 Node-to-Sink Periodic Traffic Scenario

simulation parameters	
area	300 m × 300 m
nodes	90 uniform distribution
routing	shortest path (dijkstra)
header	80 *bit*
payload	80 *bit*
simulation runs:	100
simulated time:	3600 sec

We implemented the WiseMAC *more bit* scheme and the *extended more bit* scheme and ran the same simulation of section 5.4.3 with the 90 nodes generating Poisson traffic of rate λ, and measured the common traffic characteristics at the sink.

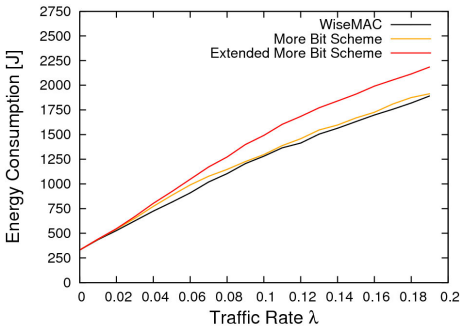


Figure 5.51: Energy Consumption

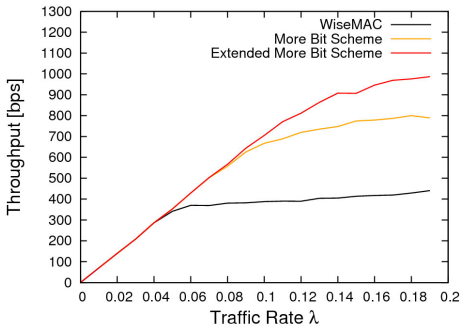


Figure 5.52: Throughput

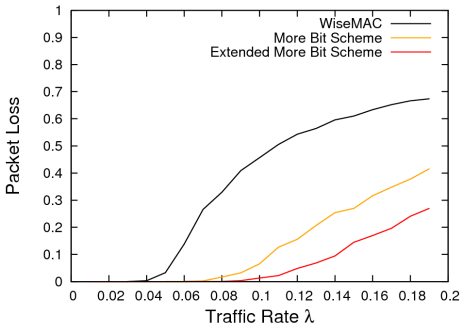


Figure 5.53: Packet Loss

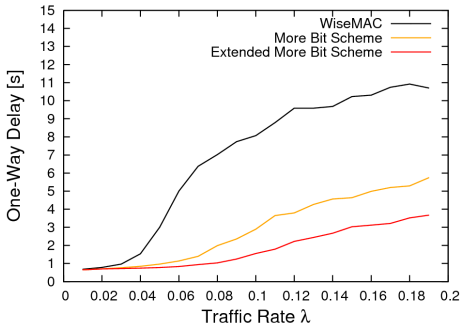


Figure 5.54: One-Way Delay

Figure 5.52 shows that indeed, an increase of maximum throughput is possible. As illustrated in figures 5.52-5.54, the extended more bit scheme accomplished by the stay awake promise proved to be superior to the WiseMAC more bit scheme in respect to throughput, packet loss and one-way delay. However, as illustrated in 5.51, the improvement comes with an increase of the overall energy consumption.
It is quite difficult to improve the WiseMAC scheme coupled with the more bit mechanism in respect to traffic adaptivity, and yet to retain the energy efficiency. When intending to provide a mechanism with least-possible energy consumption for low-traffic-scenarios, the original WiseMAC scheme seems to be already quite suitable. If throughput increase is necessary and mandatory, the figures above show that measures can be taken to increase the maximum throughput, with a considerable increase of the energy consumption.

5.5.2.2 More Bit and Extended More Bit on the ESB

We implemented the more bit scheme and the extended more bit scheme based on the *stay awake promise* on the WiseMAC prototype implementation on the ESB, and tested out the performance in two simple scenarios. Implementing the schemes was quite straightforward, signalizing to stay awake is achieved by altering a single bit in the MAC header. We faced difficulties with the memory limitation and with the system stability when increasing the load. As the memory only allows to store 5 packets, a lot of packets are dropped due to buffer overflows.

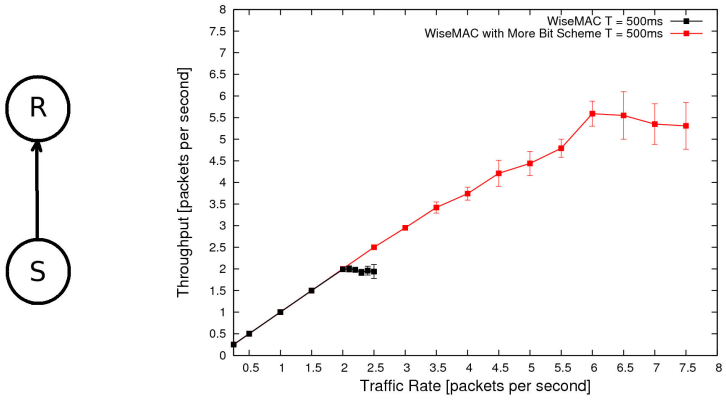


Figure 5.55: WiseMAC with and without more bit scheme on the ESB

In a first step, we tested out the more bit scheme of WiseMAC in a simple sender-to-receiver scenario over one link. We applied constant rate traffic of rate r (packets per second) with the same parameters of the prior investigations and experiments carried out with the ESB, as described in section 4.2, and measured the throughput at the receiver node. The experiment setup and results are depicted in figure 5.55.
The black line illustrates the behavior of the WiseMAC protocol without the more bit scheme. It can only deliver one packet per wake-up, and therefore, throughput is restrained to maximum two packets per second with the basic cycle duration $T = 500\ ms$. When increasing r further on, packets are subsequently queued in the buffer. When the buffer is full, packets are simply dropped. No more than one packet per wake-up can be sent, and the throughput remains below the two packets per second. This behavior is visible in the black curve between 2 and 2.5 packets per second.
With the WiseMAC more bit scheme, more than one packet can be sent with each wake-up of the receiver. The sending station receives packets from its application layer and buffers them until the receiver node's next wake-up. The sender then transmits frames with the more bit set, listens for the acknowledgement and continues sending the next packet in line, until its buffer is empty. The implementation succeeds to send $4-5$ packets in a row. Sometimes, the acknowledgement is missed or not received correctly, what leads to one long burst being segmented into two shorter bursts. Yet, by applying the more bit scheme, we could increase the throughput to much higher values. When comparing to WiseMAC without the more bit, the transmission rate could be approximately tripled.

With increasing traffic rate, the system became instable and likewise crashed during some measurement runs. The memory limitation and the fast switching between send, receive and sleep seemed to negatively impact on the system stability, as stack-overflows occurred with increased traffic generation rate. As we could observe a stagnation of the throughput at rate $r = 6$, with packet loss values varying heavily between the measurement runs, we decided to cease the investigations and measurements at this point.

In the second experiment, we tested out the two schemes *more bit* and *extended more bit* when generating traffic of equal rate in two senders S_1 and S_2 destined to one receiver R, as illustrated in figure 5.56 on the left. When both sending nodes S_1 and S_2 aim to concurrently forward packets to R, the node R becomes a bottleneck, as both nodes aim to concurrently transmit their packets during the limited wake-ups. In section 5.5.2.1, we could claim some limited throughput increase with high traffic when applying the extended more bit or *stay awake promise*, compared to the WiseMAC more bit scheme.

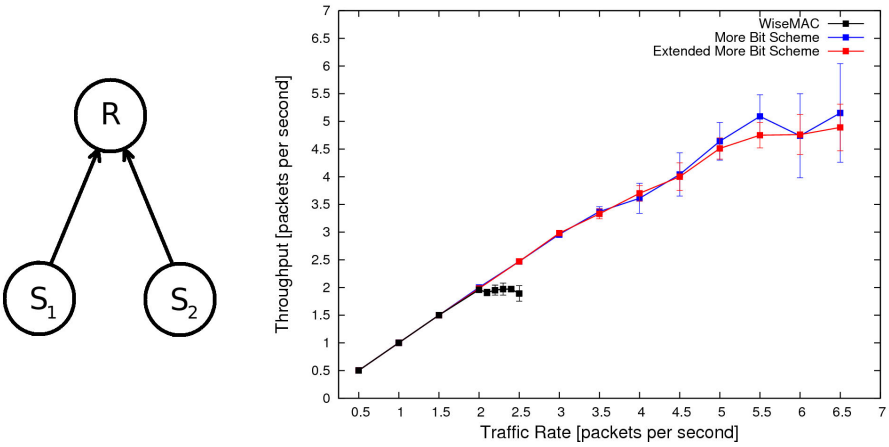


Figure 5.56: WiseMAC, More Bit and Extended More Bit

With the extended more bit scheme, the receiver node receives the more bit in the data frame header and promises to *stay awake* with a single bit in the respective acknowledgement frame. It promises to stay awake for at least $T = 500\ ms$ When receiving or overhearing the promise bit in the acknowledgment, nodes mark the time of arrival of the promise in the respective neighbor entry, such that they can later seamlessly continue sending their own packets when the neighboring node's burst is over, as illustrated in figure 5.50.

The black graph illustrates the behavior of WiseMAC without the more bit scheme, the red graph the more bit and the blue graph the extended more bit scheme. The x-axis corresponds to the aggregated traffic generated by both nodes. At the traffic of rate r in the x-axis, every sender node S_1, S_2 generates traffic of rate $r/2$. Again, the limitation of one transmission per wake-up in WiseMAC is obvious. Throughput is limited to the frequency of the wake-ups, no more than 2 packets a second can be delivered. When increasing r further on, packets are subsequently queued in the buffer and dropped when the buffer is full.

When two stations apply the WiseMAC more bit or the extended more bit scheme, they can alternately empty their transmit buffers in packet bursts with increasing traffic. The throughput reaches $4.5 - 5$ packets per second. Beyond the rate of $r = 5$, the behavior of the system again became very instable and also node-dependent. Many packets were dropped due to buffer-overflows and many measurement runs had to be restarted when nodes suddenly crashed and had to be flashed. The throughput increase and benefit of the stay-awake promise compared to the more bit scheme could not be proved on the on the ESB nodes, both schemes behaved more or less equal. The extended more bit scheme at least delivered more or less the same measurement values as the simple more bit scheme for the traffic rates that were evaluated.

5.6 Convergecast and Data Aggregation

Sophisticated data aggregation techniques in wireless sensor networks try to construct cost-minimal spanning trees over the network topology, such as the Tree-based Convergecast mechanism described in 2.5.1. The mechanism requires total topology knowledge in the root node, and introduces costly control messages.

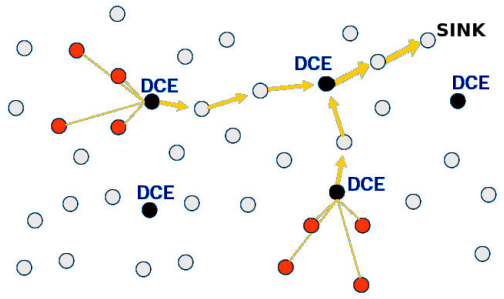


Figure 5.57: DCE nodes combining and aggregating data

We aimed to find a reasonable and cheap solution for low-traffic scenarios, which we could integrate with the flat-based low-duty cycle WiseMAC and an on-demand routing scheme, and which would further not require too heavy and complex system- and network-wide computations and communications. We found that the simple approach of the data combining entities (DCE-nodes) discussed in [29] and summarized in 2.5.2 would fit to our needs quite well. We renounced on costly mechanisms to find optimal schedules or optimal aggregation points, but sticked to an approach that is distributed, both easy to implement and requires no additional energetic cost.

We tested out the DCE-nodes approach in the OMNeT simulator on the 90 nodes topology under the assumption that DCE nodes are randomly distributed across the network, as illustrated by the black nodes in figure 5.57. Each node decides to act as data combining entity in the bootstrapping procedure with a certain probability P_{DCE}. We chose 1 *sec* as buffering interval limit and a maximum convergence limit of 5 packets meaning that a DCE node converges the application payload of incoming data packets of up to 5 packets, thereby maximally saving 4 headers. We applied the mechanism with a DCE probability P_{DCE} of 10%, 30% and 50% to the 90 nodes uniformly distributed network and the same environment parameters as outlined in sections 3.1 and 4.1. We ran the same simulation of section 5.4.3 with every node reporting data with Poisson traffic of rate λ. Again, we applied static routing to omit effects that the results are biased by the routing scheme.

simulation parameters	
area	300 m × 300 m
nodes	90 uniform distribution
routing	shortest path (dijkstra)
header	80 *bit*
payload	80 *bit*
simulation runs:	100
simulated time:	3600 sec

Node-to-Sink Periodic Traffic Scenario

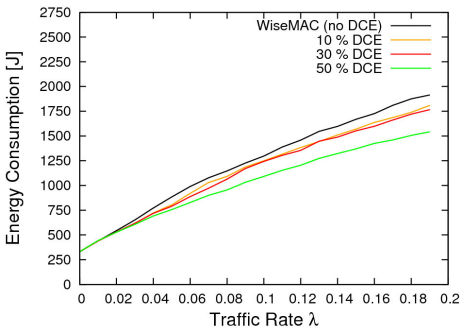


Figure 5.58: Energy Consumption

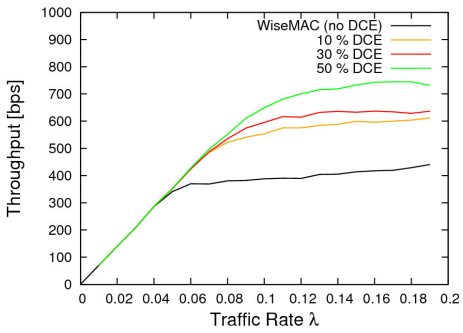


Figure 5.59: Throughput

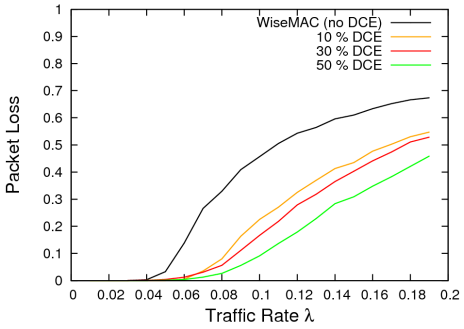


Figure 5.60: Packet Loss

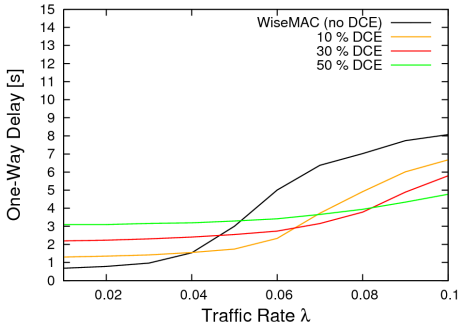


Figure 5.61: One-Way Delay

The results confirm our expectations and an increasing performance gain can be claimed in respect to the maximum achievable throughput and the energy consumption of the network. The performance gain comes with the cost of a higher end-to-end delay with low traffic-scenarios. Surprisingly, the end-to-end latency then increases less steeply. When applying the DCE nodes approach, congestion problems arise much later. The mechanism allows to keep the network operable with much higher traffic rates. This must be the effect of the lesser transmissions. Without aggregating the data in DCE nodes, every node transmits every packet alone. There are much more transmissions going on which hinder each other in forwarding data, as the increased carrier sensing range tells them that the medium is not free. With DCE nodes, nodes wait until they have gathered some packets and forward them in one big transmission. This leads to fewer interference, a more efficient medium allocation and results in lower packet loss and better throughput.

Chapter 6

Routing Layer Issues

The initial goal of the thesis consisted in the development of an integral energy-efficient MAC and routing solution for wireless sensor networks. We aimed to exploit cross-layer information passing from MAC to the routing layer to achieve performance optimizations, preying upon the special properties of MAC protocols with the individual and unsynchronized wake intervals. We indeed tackle this issue in this chapter, and find some possible optimizations by cross-layer information passing. However, the length of the chapters 5 and 4 compared to this chapter 6 makes it apparent that the investigations on the MAC - the analytical and practical improvement and evaluation of our very own suggestions to optimize unsynchronized wireless sensor MAC protocols have become the main focus and the center of attention of the thesis. Nevertheless, this chapter briefly outlines the ideas and inspirations pursued on the network layer.

This chapter discusses the efforts that were undertaken on the integration of an on-demand routing protocol to the WiseMAC protocol. We chose AODV as a well established, efficient routing protocol for different reasons.
The flat-based non-hierarchical AODV protocol with its one-hop paradigm of the routing table fits well to WiseMAC with its schedule offset table of the one-hop neighbors. AODV is both energy- and memory-efficient. It calculates routes only if they are really needed, and easily adapts to changes in the network topology. This might be of use in sensor networks, as nodes might be prone to slight mobility and move in and out of range of each other, or paths might break with forwarding nodes draining out of energy.
AODV neglects to transmit and store the full routing information between two endpoints. The route knowledge itself is distributed in the network, which makes sense in a resource-constrained wireless sensor network. The ratio between header overhead and payload yields a better energy-efficiency than DSR. AODV routing headers are considerably small, as AODV does not send the full route with every packet. Nodes do not have to maintain large route caches, a few entries for the sink stations will suffice.

Furthermore, different multipath routing variants of AODV have been proposed that offer to establish multiple paths by exploiting particular properties of the route request flooding, which consolidates our choice for AODV. In this chapter, we summarize the investigations that were undertaken on the issue of load-balanced routing to achieve a longer lifetime of the sensor network and the problems that were encountered with this approach in simulation and with the Embedded Sensor Boards prototype.

6.1 AODV Evaluation

We implemented the Ad Hoc On-Demand Routing protocol (AODV) both on the OMNeT simulator and on the Embedded Sensor Boards testbed within some simplifications. In the following, we briefly describe what features of AODV were implemented and which options were left out.

6.1.1 AODV Routing in OMNeT

We integrated the following AODV control messages on top of the WiseMAC simulation in OMNeT:

- Hello Message (HELLO)
- Route Request (RREQ)
- Route Reply (RREP)
- Route Error (RERR)

In AODV, a node offers connectivity information by broadcasting local HELLO messages. By periodically sending and receiving HELLO messages, nodes learn to know their neighbors and keep their knowledge about the local connectivity up to date. Within a bootstrapping phase of 20 secs, nodes broadcast a few HELLO messages by means of full-preamble WiseMAC broadcasts to make sure that nodes discover each other within the discovery phase. Thereafter, nodes broadcast HELLO messages only with a HELLO_INTERVAL of 100 secs, to make sure that network capacity is not eaten up entirely by HELLO messages.

A node aiming to transmit data first checks whether it knows a valid route to the destination. If not, it floods a RREQ across the network. When receiving the broadcast, an intermediate node checks if it has a *fresh enough* routing entry to the desired destination node. If it does, it generates an unicast RREP message towards the originator. AODV proposes to optionally send a so-called gratuitous RREP to the destination node to inform also the destination about this route. Our implementation omits this feature, as later incoming AODV data packets fulfill this purpose anyway.

If a link failure is recognized, AODV offers two options. The first option is to send a RERR to inform the other nodes about that link failure and the second option is to try to repair the link by generating RREQ to the affected nodes (so called local repair). We decided to stick to the first approach. When a node receives a data message for a destination it does not have a valid route for, it will return a RERR. The RERR will be unicast towards the originator, and erases the respective failing route to the desired destination in every intermediate node.

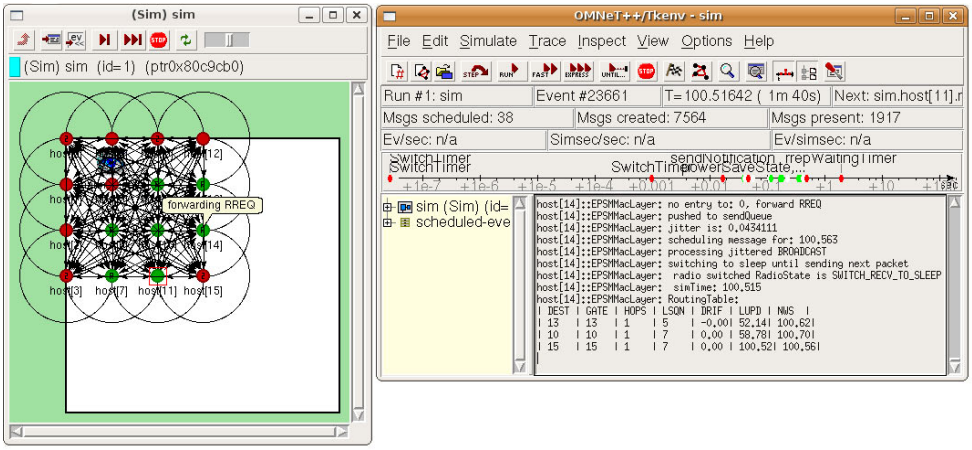


Figure 6.1: OMNeT Simulator with WiseMAC/AODV

6.1.2 AODV on the Embedded Sensor Boards

```
struct routing_entry_struct {
        UINT16 destAddr;
        UINT16 destSeqNr;
        UINT16 nextHop;
        UINT16 hopCount;
        UINT32 installTime;
};

struct rreq_struct {
        UINT16 sourceAddr;
        UINT16 sourceSequenceNr;
        UINT16 broadcastId;
        UINT16 destAddr;
        UINT16 destSequenceNr;
        UINT8  hopCount;
};

struct rrep_struct {
        UINT16 sourceAddr;
        UINT16 destAddr;
        UINT16 destSequenceNr;
        UINT8  hopCount;
};

struct hello_struct {
        UINT16 sourceSequenceNr;
        UINT8  force;
};
```

Figure 6.2: AODV control message structs in the ESB prototype

We improved and adapted previous work on the AODV routing protocol for the Embedded Sensor Boards contained in version 3.2 of ScatterWeb [42] and integrated it into our WiseMAC prototype implementation. Figure 6.2 lists the AODV table entry structs as well as the structures used to implement the AODV control messages RREQ, RREP and HELLO.
We were confronted with different serious problems when implementing the main features of the AODV protocol on the ESB nodes. It turned out that the memory limitation was a massive restriction for the programming task. The ESB nodes feature only 2 kbyte RAM and 60 kbyte ROM. Out of the 2 kbyte, ScatterWeb uses approximately two thirds for send and receive buffers and other variables (i.e. timers, variables, configuration settings). We had already used quite much memory for the WiseMAC implementation where we had needed large arrays for the schedule offset table of the neighbor entries, countless variables for the implementation and some buffers for the preamble duration calculations and k-best-instants related calculations. We therefore had to limit the MAC layer to have at maximum 4 neighbor entries and the AODV table to maximum 3 entries. The same problems were encountered for the text segment that contains the code. Out of the 60 kbyte ROM, the ScatterWeb code already allocates approximately 55 kbyte. We removed all parts of the ScatterWeb code that seemed unnecessary for our tasks, i.e. all all commands and all instructions for accessing and reading out sensor data from the infrared, the temperature and the vibration sensors. Still, the memory limitation imposed a massive restriction. We had to keep the implementation as small as possible. We decided to only implement the key features for bidirectional route establishment and data transfer and neglected to implement features such as the local repair, route error or route maintenance.
Finally, we managed to run the main features of the AODV protocol on top of the implementation of the WiseMAC prototype, which allowed to make some small-scale experiments.

Route Establishment Experiment

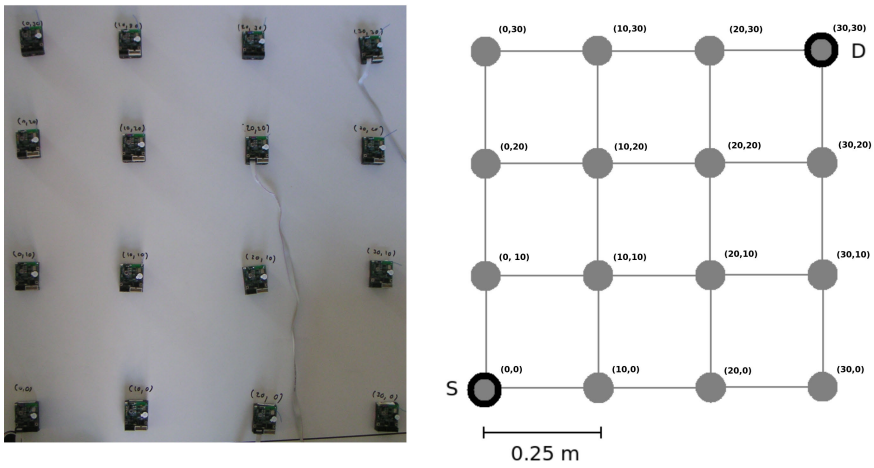


Figure 6.3: Lattice Square Topology with ESB nodes

We lay out out 16 nodes in a 4x4 lattice square topology, as depicted in 6.1.2. We assigned positions to the nodes. Position (0,0) was assigned to the node in the lower left corner. For each hop in each direction the positions were likewise increased by 10. Position (30,30) was therefore assigned to the node in the upper right corner. We set the *virtual range* parameter MAX_RANGE of of the MAC filter methodology outlined in 3.2.4 to 10. Nodes are all in complete range of each other, but packets from nodes that are too far are filtered out. Only the links depicted in the graph of Figure 6.1.2 are actually used. A node therefore has 4 neighboring nodes at maximum.

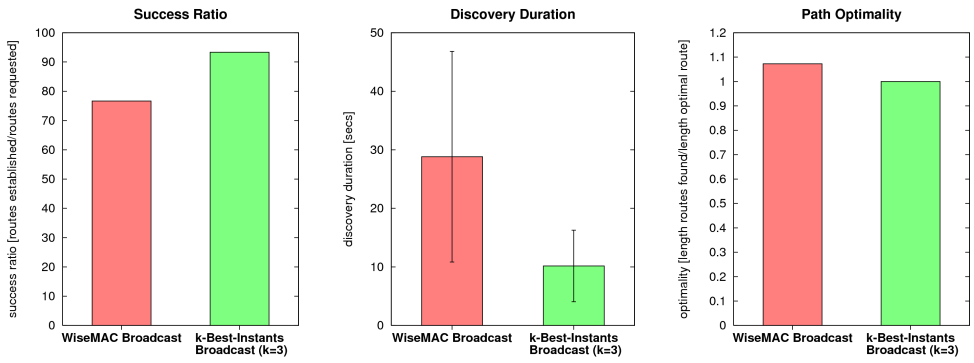
We triggered 30 Route Requests (RREQ) from the source node S in the lower left corner (0,0) to the destination node D in the upper right corner (30,30), and measured

- *if* the request succeeds and a route is found to the destination (success ratio)
- *how long* it takes from the first emitted RREQ broadcast to the first incoming RREP (path discovery duration)
- *path optimality* as ratio of the length of the path found compared to the length of the optimal route

The results give an insight how robust the implementation of the routing protocol works. It can be seen as a proof of concept that the work on the WiseMAC prototype in sections 4.2 and 5 can be successfully deployed in a multi-hop wireless sensor network operating with an on-demand routing protocol. Given the opportunity to test out the suggested broadcast scheme of section 5.3, we ran the experiments with both, WiseMAC full-preamble broadcast and k-best-instants broadcast technique to compare and analyze the respective behavior and robustness.
We figured out the parameters that were most appropriate and delivered the highest success ratios for each broadcast variant. The parameter RREQ_JITTER denotes the jitter between incoming RREQ and the rebroadcast and is a random value in the given interval. The parameter RREQ_RETRY_TIMER denotes the time between the first RREQ broadcast of the originator node that searches a path and its first retry attempt.

Parameter	WiseMAC Broadcast	k-Best-Instants Broadcast
HELLO_INTERVAL	500 s	500 s
RREQ_RETRIES	3	3
RREQ_RETRY_TIMER	15	10
RREQ JITTER	uniform [0,3000] ms	uniform [0,1000] ms

We had to choose higher values for the RREQ JITTER and the RREQ_RETRY_TIMER when applying the WiseMAC full-preamble broadcast. When choosing lower values, collisions occurred and hindered the RREQ from spreading across the network, which led to starvation of the flood. The reason is in the most cases the ScatterWeb carrier detection, that is still not very reliable. As the carrier detection merely relies on the recognition of startbytes, and the preamble signal is not always recognized correctly, nodes often misinterpret the medium to be *idle* when in fact it is not. Nodes then forward a RREQ and access the medium for transmission, not detecting already ongoing transmissions and interferring with them, what leads to packet losses due to collisions. When choosing a higher jitter and a higher retry timer, we could lower the probability that collisions occur, and increase the chance that a route request actually could get through and reach the destination node D. The results we then obtained are depicted in the bars below.



We measured a success ratio of roughly 76.66% with the WiseMAC broadcast approach and a success ratio of 93.33% with the k-best-instants technique. These success ratios prove our concept that the very energy efficient WiseMAC power saving scheme can be quite reliably applied in multi-hop sensor networks based on on-demand routing.
The results furthermore delivered better results with the k-best-instants approach in respect to success ratio, path discovery duration and optimality than the WiseMAC full-preamble broadcast. However, the interpretation of this gap is according to our observations, a direct consequence of the unreliable ScatterWeb carrier detection. As the carrier detection is unreliable to a certain degree, it is likely that a node will wrongfully consider the medium to be idle when it actually is not during the very long full-preambles of the WiseMAC broadcast. Simultaneously transmitting stations will most likely interfer with each other and thus hinder the broadcast from spreading across the network. With the k-best-instants broadcast, forwarding incoming RREQ's blocks the medium only for the multiple short transmissions, so only for roughly 50 ms. The danger that rebroadcasts collide with each other and hinder the RREQ from spreading across the network is then much lower.

The satisfactory conclusion remains that the AODV implementation managed in most cases to find a route between the most distant nodes in the network, which is 6 hops away. As the thesis already experiments with traffic along a linear chain in chapter 4 and 5, we omitted to do so in this part of the thesis, as no new insights can be expected.

6.2 Multipath Routing

In this section, we outline the efforts that were undertaken to integrate a load-balancing multipath routing scheme into the WiseMAC simulation in OMNeT, and to the ESB prototype, with the focus of increasing the sensor network lifetime. We begin with the implementation of a multipath extension of the AODV protocol on top of WiseMAC in OMNeT, with which we failed to prove performance gains. We then slightly altered and extended this implementation with own policies and an approach to exploit MAC layer information on the routing layer, until we could claim some theoretic performance gains on the simulator. We finally discuss the challenges we met to achieve lifetime improvements in sensor networks that apply WiseMAC on the MAC layer, and the attempt to implement the load-balancing approach on the ESB nodes.
We investigated on techniques to obtain more than one route in a cheap manner, preferably by exploiting duplicate RREP's of an initial RREQ query. Two multipath extensions of AODV for finding multiple routes in one RREQ-RREP query cycle have been recently proposed, called AOMDV [26] and AODVM [27].

We found that the AOMDV approach described in 2.3.3.2 is applicable with WiseMAC. The second extension AODVM relies on intermediate nodes overhearing RREP's that are subsequently sent back to the originating node, and bookkeeping of the transmissions of the intermediate nodes to ensure node-disjointness. Overhearing of intermediate node's transmissions can unfortunately not be guaranteed when applying the WiseMAC protocol on the MAC. Nodes have their radio only turned on for very small duty cycles. They might overhear some transmissions, but certainly not all of them.
We therefore sticked to the AOMDV approach, which does not rely on the property of overhearing. AOMDV finds node-disjoint paths by exploiting a particular property of flooding, as described in section 2.3.3.2. We applied the route discovery mechanism to find only the node-disjoint paths of AOMDV, leaving out the option to discover more link-disjoint routes. According to the AOMDV route update rule, a each routing entry is only added if the *first hop* is different than the ones of prior received duplicate RREQ's. The scheme succeeded in finding multiple paths in some cases, but the results in respect to network lifetime became even worse than pure AODV routing, approximately 10% in respect to the achieved lifetime. Even when weighting the transmissions over suboptimal paths with a lower probability, such that these routes would only be taken occasionally into account, as proposed by *Ganesan et al.* in [24] and described in 2.3.3.5, the performance resulted slightly worse than pure AODV.
It turned out that the redundant paths were often much longer than the optimal paths. We assume that the long detours of redundant paths impacted negatively on the lifetime, as more transmissions become necessary when paths are suboptimal, and each transmission may influence other nodes in the large carrier sensing range. We therefore ceased to follow this strategy and renounced to elaborately examine the performance of this approach.

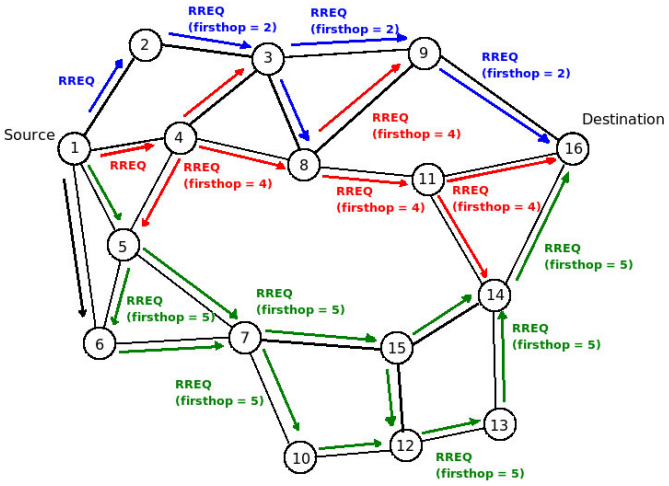
We began altering the table update policies and observed what happens if we changed one or another condition. The upcoming section discusses the modifications that led to a slightly better performance on the simulation platform.

6.2.1 AODV- and AOMDV-inspired Multipath Approach

6.2.1.1 Altering the Table Update Policy

As AODV is tailored for the use in mobile ad hoc networks, it always keeps the *freshest* route to every destination. A node receiving a path advertisement for a given destination node checks if the advertisement provides a higher destination sequence number, or if it provides an equal destination sequence number *and* a shorter path. If it does, the current entry for this destination is deleted and the packet source is taken as new gateway for the destination node. As AODV has been designed for use in MANETS where nodes move and get in and out of range of each other, the sequence number condition ascertains that a node always uses the path known to be the *freshest*. Wireless sensor networks can however be assumed quasi-static, and node mobility does not play a major role. We therefore softened the condition of prioritizing route advertisements with highest sequence number. Our approach considers route advertisements to a destination with higher sequence number only if the route is not longer than the current. The approach incorporates the basic mechanism of the AOMDV protocol to find node-disjoint paths, but restricts to add such paths only if they advertise the same hopcount. Incoming RREQ duplicates are treated as in AOMDV, they are answered if they advertise a node-disjoint path to a destination but only if the advertise the same hopcount. We add an additional path entry to the same destination to which a path is already known if:

- *a)* the sequence number is equal or higher
- *b)* the first-hop is different from all already known paths to the same destination
- *c)* the hopcount is equal



AODV

routing table node #1

dest	next	hops	seq
16	2	4	11

routing table node #16

dest	next	hops	seq
1	9	4	23

AOMDV

routing table node #1

dest	next	hops	seq
16	2	4	11
16	4	4	11
16	5	6	11

routing table node #16

dest	next	hops	seq
1	9	4	23
1	11	4	23
1	14	6	23

only optimal routes

routing table node #1

dest	next	hops	seq
16	2	4	11
16	4	4	11

routing table node #16

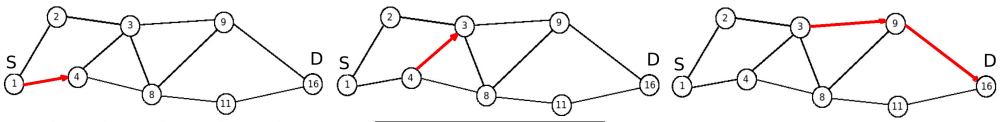
dest	next	hops	seq
1	9	4	23
1	11	4	23

Figure 6.4: RREQ and different table update policies

When a path-advertisement arrives with lower hopcount, all existing routes are deleted and the new route is added. When receiving a duplicate that fulfills the conditions of node-disjointness of AOMDV *and* is optimal in terms of hop count, the routing table is extended to contain more than one path entry.
The routing table entry update rule modification in comparison with AOMDV and AODV can be explained with figure 6.4, where the dissemination of a route request from node 1 searching a path to node 16 is depicted. After flooding the whole network, the destination node receives path advertisements to node 1 from its neighbors 9, 11 and 14.
With AODV, the destination node only answers to the first incoming RREQ with a corresponding RREP, let it be from 9. The duplicate RREQ from 11 is simply discarded and left unanswered, as it advertises the same sequence number. Although it took another route and would provide path redundancy, AODV discards the request and leaves it unanswered. In contrast, AOMDV considers *all* routes that are advertised by 9, 11 and 1, as the respective RREQ's all took another first hop.
We altered the table update policy such that only the hop-count optimal routes are added to the table and answered with a RREP. In the upper case, the RREQ's received over 9 and 11 are answered with a RREP, but not the one over 14. The resulting routing tables for the source node 1 and the destination node 16 are depicted figure 6.4. With AODV, only one path entry is considered, whereas AOMDV adds all paths to its table. With our approach, only the hop-count optimal routes over 9 and 11 are added to the table.

6.2.1.2 Shortest Inter-Delay Routing

AOMDV only topics the question how to establish multiple routes, but not how to spread the load over them. There are probabilistic schemes which assign a certain probability to a route and choose the route for each packet in a random manner. We suggest to exploit information provided by the MAC layer to achieve some performance gains in respect to the latency. As all redundant path entries to a destination advertise an optimal route in terms of hop count, the next soonest wake-up of the gateway leading to the destination shall be the only selection criterion, also in each intermediate node. For a transmission of a packet from source to destination, each intermediate node shall forward the packet to the gateway of the destination with the soonest wake-up. A lower latency as well as the desired load balancing among the intermediate nodes can thus be expected.



dest	next	hops	Δt [ms]
16	4	4	132
16	2	4	54

routing table node #1

dest	next	hops	Δt [ms]
16	3	3	72
16	8	3	213

routing table node #4

dest	next	hops	Δt [ms]
16	9	2	13
16	8	2	47

routing table node #3

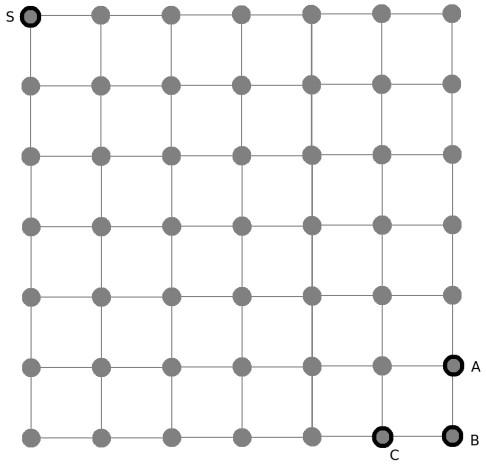
Figure 6.5: packet forwarded from 1 to 16 choosing the soonest wake-up gateways

As the source knows two paths towards node 16, it chooses the path according to the inter-delay to the next-wake-up of the gateway node. In the first figure, we can see that the time remaining to the next wake-up of node 4 is in $\triangle t = 132\ ms$, and the next wake-up of node 2 is in $\triangle t = 54s$. Therefore, the source node chooses to send the packet over node 2, as it can deliver the packet and empty its buffer earlier. Accordingly, the packet is routed in every intermediate node. As we only added hop-count-optimal routes, packets are never routed *away* from the destination.

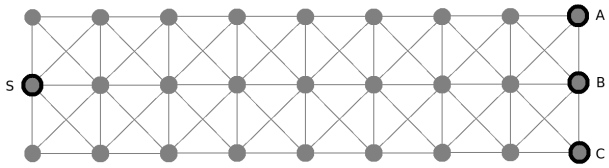
6.2.1.3 Simulation and Evaluation

Topology Setup As the choice of the network topology may have an impact on the results, we considered the following three network topologies:

- uniformly distributed network topology of 90 nodes (see section 5.4.3)
- 3x10 nodes grid topology



- 7x7 nodes lattice square topology



Lifetime Metric We defined the lifetime of the network as the time until 10% of the nodes deplete *or* the network becomes partitioned, and in a second run as the time until the first node depletion.

Traffic Patterns For each topology setup, we measured two different traffic patterns.

- *Evenly distributed traffic*: Every node starts reporting data according to the Poisson model with $\lambda = 0.01$. When every node generates the same amount of traffic, multipath routing might not pay off, as the load is already balanced. As common single path routing protocols establish source-sink trees with some nodes having the burden to forward traffic of large subtrees, multipath routing still might help to redistribute the load over more and other hops.
- *Neuralgic spots traffic*: If there are neuralgic spots in the network that generate much traffic, whereas other parts stay more or less inactive, multipath routing can pay off more. We assume that the three most distant nodes from the sink generate 20 times more traffic ($\lambda = 0.05$) than all other nodes ($\lambda = 0.0025$).

Lifetime Results

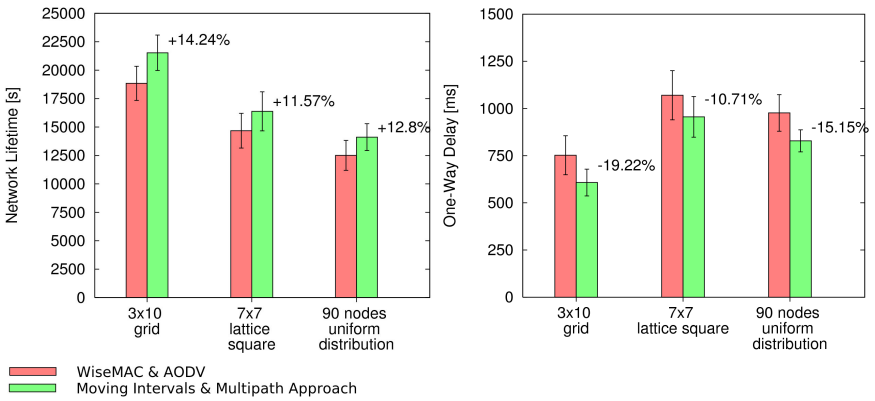


Figure 6.6: Network Lifetime and One-Way Delay with Evenly distributed traffic

The results show an overall performance gain when applying the AOMDV-related scheme coupled with the next-soonest-wake-up routing paradigm of ∼10-15% concerning the network lifetime and the one-way delay. When considering the low cost of some additional RREP messages in the initial route discovery flood, the results state that on-demand multipath routing may provide a limited but valuable contribution to prolong the network operability. In our simulation, the mechanism only paid off when sticking to the hop-count-optimal routes only.

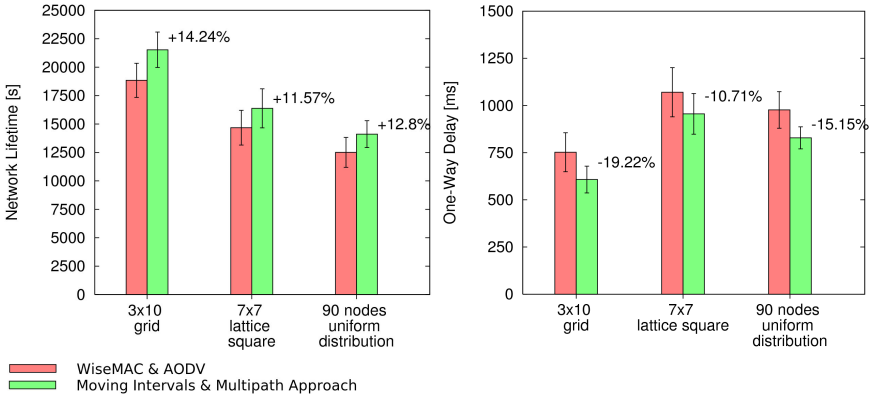


Figure 6.7: Network Lifetime and One-Way Delay with Neuralgic spots traffic

The exploitation of the MAC-layer information about the next-soonest wake-up of the neighboring nodes paid off in respect to the one-way delay. This might stand opposite to the OSI design paradigm of least possible coupling between the layers, but in a wireless sensor networks with scarce energy resources, such measures are acceptable when they serve a higher purpose.
When comparing figures 6.6 and 6.7, we conclude that neither the different traffic patterns nor the topology setup has a vast influence of the results. More or less the same performance gains could be claimed with each scheme.
In a second run, we ran all the experiments with the lifetime metric of the first node depletion. However, the results differed only slightly from the results depicted in the

upper figures. The overall lifetime gain also reached 10-15% in respect to the lifetime and the one-way delay for all topologies and both traffic patterns.

6.2.2 Challenges of Multipath Routing with WiseMAC

The WiseMAC protocol introduces an extended carrier sense range in order to mitigate the hidden node problem. A low carrier sense threshold on the measured received signal strength lets nodes consider the medium *busy* much earlier than in other wireless channel protocols, where the carrier sensing range equals the transmission range. This more prohibitive carrier access policy makes it much much more probable that routes do interfer with each other. The extended carrier sensing range leads to much more potential interference with the transmissions of other nodes, even if the routes do not share a common link or node.

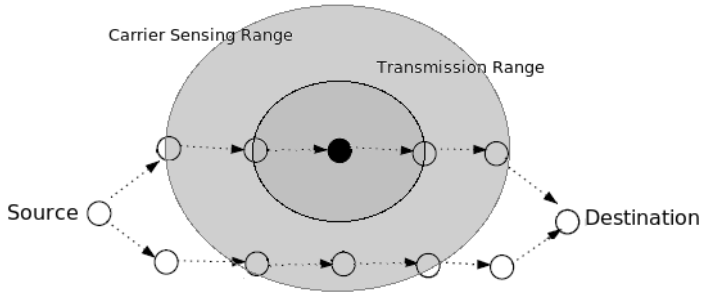


Figure 6.8: Route Coupling with increased Carrier Sensing Range

The impact of *Route Coupling*, as discussed in 2.3.2, is much bigger with WiseMAC. In figure 6.2.2, nodes form a route without any connection in between. The two routes are node and link-disjoint. When extending the carrier sensing range, two or more nodes of the lower path fall into the reach of each node in the upper path, and vice-versa. When one node of the upper path is transmitting, the nodes of the lower path will have to remain silent, as they recognize the medium to be *busy*. When a node aims to transmit a frame to the next node in the path, the probability that it cannot access the medium due to the concurrent transmissions of nodes in its carrier sensing range is much higher than in the single path case. Spreading the traffic over multiple routes when applying an increased carrier sensing range leads to much harder route coupling and interference problems.

We analyzed and observed the network functioning when applying different table update policies and different traffic patterns. We find that the route coupling problem is responsible for the problems encountered in sections 6.2.1.1 and that the results of the multipath approach remained below our expectations in general. With the much bigger carrier sensing range, the danger that transmissions along multiple paths interfer with each other is much higher and the load balancing effect is soon exceeded by the additional cost of coping with the path interference. This is the essential insight gained during the work pursued in this chapter. The impact of the extended carrier sensing range on the route coupling problem could be subject to future research.

6.2.3 Multipath Routing on the Embedded Sensor Boards

We implemented the AOMDV mechanism to find multiple node-disjoint paths based on the *first-hop* field in the RREQ on the ESB nodes, and aimed to find multiple routes in the 4x4 nodes lattice-square topology network of section 6.2.
We faced several difficulties to make useful measurements with the experiment setup and the methodology available. It turned out that it is not possible at all to measure the lifetime of the sensor network with the methodology of the GoldCap capacitors, as we can only power one single node with a capacitor and not the entire network. The network initialization phase itself already needs some 200 seconds until every node has found its respective neighbors, which is more or less the energy a node can live of the initial charge of the capacitors.
After all, it turned out that the AOMDV route discovery scheme failed to deliver two node-disjoint routes with sufficient probability. In some experiment runs, the scheme indeed delivered two routes, but we neglected to make actual measurements of the one-way delay of packets routed over multiple paths as this case was too rare. AOMDV does not guarantee to find node-disjoint paths, even if the topology would allow to find some. In order to find two node disjoint paths, the RREQ must spread over both first-hop nodes of the source node with position (0,10) and (10,0) in two concurrent waves and each RREQ wave must reach one neighbor of the source. The RREQ over node (0,10) would have to reach the first neighbor of the destination (20,30) and the RREQ over (10,0) would have to reach the second neighbor of the destination (30,20). If this does not succeed, and the RREQ does not spread equitably in two waves over the network, node disjoint paths are not found. As this case even proved to happen only occasionally on the simulator, we ceased our investigations on multipath routing on the ESB, and invested our time on further investigations on the MAC.
According to our observations, the inherently unreliable ScatterWeb carrier detection is accountable that in most cases, only one RREQ makes it to the destination, as this RREQ wave suppresses all other RREQ transmissions. The unreliable carrier detection lets nodes often misinterpret the medium to be *idle* when in fact it is not, whereafter they begin transmitting and interferring other nodes' RREQ transmissions.

Chapter 7

Conclusions and Outlook

This thesis combines features and ideas of previous work on unsynchronized MAC protocols for wireless ad hoc and sensor networks, mainly the basic concepts of the WiseMAC [4] and the Unsynchronized Fixed and Random Intervals protocol [2] protocol. In contrast to a huge amount of energy-saving MAC protocols, unsynchronized power saving MAC protocols suggest to renounce on any kind of network-wide or clusterwise synchronization for the channel access coordination, as it is done in many scheduled protocols, nor for the coordination of a common wake and sleep pattern. They neglect to exchange costly MAC layer control messages in the belief that for wireless sensor networks with low traffic requirements, maintaining a multi-hop synchronization scheme to control slotted medium access is too costly. The thesis aims to design an integral medium access control and routing mechanisms solution tailored for use in wireless sensor networks renouncing on costly synchronization measures.

The implementation of the unsynchronized power saving protocol WiseMAC forms the entry point for the thesis' investigations on the MAC. The implementation of the preamble sampling technique basing on least-possible duty cycles necessites a very exact and precise exchange and bookkeeping of the schedule information about the neighboring nodes. The WiseMAC prototype permits to lower the duty cycle to 1% or 5 ms every $T = 500$ ms, but still offering reasonable connectivity properties. The WiseMAC implementation of this thesis is quite likely the most energy-efficient power saving MAC protocol for low-rate traffic scenarios currently implemented on the ESB research platform. The thesis discusses the performance of the WiseMAC protocol in simulation and in practice. Cross-comparisons of the respective investigations' results yield the opportunity to discuss the choice of appropriate and realistic parameter values for the simulator model, with which similar results are obtained both on the simulator and on the sensor hardware platform. The thesis demonstrates how the choice of parameters impacts on the obtained results and that careful investigations on simulation parameters pay off when aiming to prove simulated effects in practice.

The thesis suggests performance optimizations considering the broadcast operation mode to achieve a higher energy-efficiency both at the sender and the receiver. It carries out experiments to quantify the energy-saving effect in simulation and in practice on a sensor hardware testbed. The experiments approve the energy-efficiency of the scheme further called *best-instants broadcast* when a limited amount of neighbors has to be reached. The thesis further tests out this broadcasting technique in on-demand routing schemes both in simulator and on a sensor hardware testbed and advices especially to apply this technique whenever multi-hop network-wide flooding is required. The later implementation of an on-demand routing scheme approves that the technique leads to a lower medium utilization and fewer interference problems.

An alternative allocation and arrangement scheme of the node wake-up's is discussed to avert performance degrading systematic overhearing and fairness effects of the WiseMAC's fixed static wake-up pattern, as problems may arise when two neighboring nodes share a

similar wake patterns. The thesis suggests a scheme to let the node's wake-up intervals move in inside fixed cycles in respect to a linear movement-function. The thesis models a proof that this scheme leads to a more reliable overhearing avoidance and a lower danger of systematic overhearing than the WiseMAC fixed wake-up pattern, yet retaining the deterministic nature of the wake-ups. Different variants are tested in simulation, and the advantages of one variant is proved in practice on the sensor testbed implementation.

The thesis outlines a mechanism to improve the traffic-adaptivity of the WiseMAC protocol in cases of traffic between multiple senders and one receiver basing on a so-called *stay-awake promise*. Cases with multiple nodes aiming to forward data over certain receivers are likely to occur in wireless sensor network topologies. Bottleneck nodes often have to forward data of large subtrees. Experiments on the simulator approve the performance gain in respect to throughput and latency. The scheme is ported to the sensor testbed implementation where it succeeds in increasing the throughput in comparison with WiseMAC, but fails to prove a higher throughput in comparison with the WiseMAC fragmentation scheme *more bit*, as the effect pays off only with very high traffic rates, which are yet unsupported in the implementation.

On the routing layer, the thesis integrates the on-demand routing protocol AODV and tests out the performance of the combination with the energy saving MAC. Efforts are undertaken to integrate a multipath-protocol to balance load in a wireless sensor network and claim a higher network lifetime. Problems are encountered, as the properties of WiseMAC with the extended carrier sensing range do not favor the application of multipath-protocols. Route coupling problems arise earlier and have a stronger impact on the performance when applying a more prohibitive carrier access policy to mitigate the hidden node problem. By exploiting cross-layer optimizations between MAC and routing, and altering path update policies, the thesis finds a scheme that delivers slightly higher lifetimes on the simulator. The thesis however fails to prove the efficiency gain on the sensor hardware testbed.

Further work on the MAC could topic the question how to analytically express and quantify the overhearing avoidance effect of the moving wake intervals. Further advantages, or drawbacks and limitations of this scheme might be found when investigating on this issue in a solid mathematical manner.
On the routing layer, further investigations could topic the question how the properties of the unsychronized MAC and the moving intervals could be exploited in cross-layer approaches, i.e. to find paths with the least-possible delay when gathering the information about the wake-patterns of not only the one-hop neighbors, but all nodes between a station and the sink. Gathering more information about the neighborhood could furher help to construct multiple paths that do not interfer with each other. Nowadays established on-demand multipath routing protocols for ad-hoc networks do not account for the problem of route coupling at all. The introduction of path construction algorithms that set up maximally interference free paths and redundant paths towards one or a few sinks in an initial deployment phase might likely pay off when aiming to keep a wireless sensor network operable for weeks, months or even years. Furthermore, the correlation between the route coupling problem and the extended carrier sensing range collision avoidance scheme could be subject to future investigations.

Bibliography

[1] Mark Weiser: *The Computer for the Twenty-First Century* Scientific American, 1991

[2] Braun T., Feeney L.M.: *Power Saving in Wireless Ad hoc Networks without Synchronization*, 5th Scandinavian Workshop on Wireless Ad-hoc Networks, 2005

[3] Hurni P., Braun T., Feeney L.M.: *Simulation and Evaluation of Unsynchronized Power Saving Mechanisms in Wireless Ad Hoc Networks*, WWIC 2006

[4] El-Hoiydi, A.; Decotignie, J.-D.: *WiseMAC: An Ultra Low Power MAC Protocol for Multihop Wireless Sensor Networks*, ALGOSENSORS, 2004

[5] El-Hoiydi, A.: *Energy Efficient Medium Access Control for Wireless Sensor Networks*, PhD Thesis, École Polytechnique Fédérale de Lausanne, 2005

[6] Ye, W., Heidemann, J., Estrin, D.: *An Energy Efficient MAC protocol for Wireless Sensor Networks*, IEEE Infocom, 2002

[7] Dam, T.V., Langendoen, K.: *An Adaptive Energy Efficient MAC Protocol for Wireless Sensor Networks*, ICM Conference on Embedded Network Sensor Systems (SenSys), 2003

[8] Demirkol, I., Ersoy, C., Alagoz, F.: *MAC Protocols for Wireless Sensor Networks: A Survey*, IEEE Communications Magazine, 2006

[9] Xu, K., Gerla, M., Bae, S.:*How Effective is the IEEE 802.11 RTS/CTS Handshake in Ad Hoc Networks*, IEEE Globecom, 2002

[10] Weinmiller, J., Woesner, H., Ebert, J., Wolisz, A.:, *Analyzing the RTS/CTS Mechanism in the DFWMAC Medium Access Protocol for Wireless LANs*, IFIP TC6 Workshop Personal Wireless Communications, 1995

[11] Ye W., Silva F., Heidemann J.: *Ultra-low duty cycle MAC with scheduled channel polling*, ICM Conference on Embedded Network Sensor Systems (SenSys), 2006

[12] Hurni, P., Braun, T.: *Improving Throughput in WiseMAC* IEEE/IFIP Conference on Wireless On demand Network Systems and Services, 2008

[13] Perkins, C.E., Bhagwat P.: *Highly dynamic Destination Sequenced Distance Vector routing (DSDV) for mobile computers*, ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, 1994

[14] Perkins, C.E., M. Belding-Royer E.: *Ad hoc On-Demand Distance Vector (AODV) Routing*, IETF Internet draft RFC 3561, 2003

[15] Johnson D.B.: *Routing in Ad Hoc Networks of Mobile Hosts*, Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, 1994.

[16] Al-Karaki, J. N., Kamal A. E.: *Routing techniques in wireless sensor networks: a survey*, IEEE Wireless Communications Volume 11, 2004

[17] Dulman S., Nieberg T., Wu J., Havinga P.: *Trade-off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks*, WCNC Workshop, 2003

[18] Mueller, S., Ghosal D.: *Multipath Routing in Mobile Ad Hoc Networks: Issues and Challenges*, MASCOTS Tutorials, 2003

[19] Pearlman M. R., Haas Z. J., Sholander P., Tabrizi S. S.: *On the impact of alternate path routing for load balancing in mobile ad hoc networks*, ACM international symposium on Mobile ad hoc networking & computing. Boston, 2000

[20] Voigt Th., Dunkels A., Braun, T.: *On-demand Construction of Non-interfering Multiple Paths in Wireless Sensor Networks* 2nd Workshop on Sensor Networks *Informatik 2005 Bonn*, 2005

[21] Waharte, S., Boutaba, R.: *Totally Disjoint Multipath Routing in Multihop Wireless Networks*, IEEE International Conference on Communications, 2006

[22] Ganjali, Y., Keshavarzian, A.: *Load Balancing in Ad hoc Networks: Single-path Routing vs. Multi-path Routing*, IEEE Infocom, 2004

[23] Karl H., Willig A.: *Protocols and Architectures for Wireless Sensor Networks* Wiley, 2005

[24] Ganesan D., Govindan R., Shenker S., Estrin D: *Highly Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks*, Mobile Computing and Communications Review, 2001

[25] Lee, S.-J., Gerla, M.: *Split Multipath Routing with Maximally Disjoint Paths in Ad Hoc Networks*, IEEE International Conference on Communications, 2001

[26] Marina, M. K., Das S. R.: *On Demand Multipath Distance Vector Routing in Ad hoc Networks*, IEEE International Conference on Network Protocols, 2001

[27] Ye, Z, Krishnamurthy S.V., Tripathi S.K.: *A Framework for Reliable Routing in Mobile Ad Hoc Networks*, IEEE Infocom, 2003

[28] Shah R. C., Rabaey J.: *Energy Aware Routing for Low Energy Ad Hoc Sensor Networks*, IEEE Wireless Communications and Networking Conference WCNC, 2002

[29] Srivastava, M.B., Schurgers, C.: *Energy Efficient Routing in Wireless Sensor Networks*, MILCOM, 2001

[30] Eades, P., Whitesides, S.: *The Realization Problem for Euclidean Minimum Spanning Trees is NP-hard*, ACM Symposium on Computational Geometry, 1994

[31] Annamalai V., Gupta S.K.S., Schwiebert L.: *On tree-base convergecasting for wireless sensor networks*, IEEE Wireless Communications and Networking Conference, 2003

[32] K. Kalpakis, K. Dasgupta, P. Namjoshi: *Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks*, ACM Computer Networks, 2003

[33] Ni, S.-Y., Tseng, Y.-C., Chen Y.-S., Sheu J.-P.: *The Broadcast Storm Problem in Mobile Ad Hoc Net-works*, ACM MOBICOM, 1999

[34] Varga, A: *The OMNeT++ Discrete Event Simulation System (http://www.omnetpp.org)*, European Simulation Multiconference, 2001

[35] Mobility Framework for OMNeT++ (http://mobility-fw.sourceforge.net)

[36] Faria, D.B.: *Modeling Signal Attenuation in IEEE 802.11 Wireless LANs*, Technical Report TR-KP06-0118, Stanford University, 2005

[37] L. Kleinrock, Tobagi F.: *Packet switching in radio channels: Part I - Carrier sense multiple access modes and their throughput delay characteristics*, IEEE Transactions on Communications, 1975

[38] Feeney L.M., Nilsson, M.: *Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment*, IEEE Infocom, 2001

[39] Percival, D. B.: *Modelling of Clock Behaviour*, GALILEO TIME (IWGT), 2004

[40] Weisstein, E. W.: *Uniform Sum Distribution*, From Wolfram MathWorld (http://mathworld.wolfram.com/UniformSumDistribution.html)

[41] SaRonix *32.768 kHz Tubular Crystal NTF3238 / NTF3226 Series* Datasheet

[42] *Scatterweb: Platform for self-conguring wireless sensor networks* (http://www.scatterweb.net)
Scatterweb: User Guide and First Steps Guide (http://cst.mi.fu-berlin.de/projects/ScatterWeb/documentation/)

[43] Schiller, J. Liers, A. Ritter, H. Winter, R. Voigt, T.: *ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing*, 38th Annual Hawaii International Conference on System Sciences, 2005

[44] RF Monolithics: *Datasheet for TR1001 868.35 MHz hybrid transceiver* (http://www.rfm.com/products/data/TR1001.pdf)

[45] Texas Instruments: *Users guide for the micro controller MSP430F149* (http://focus.ti.com/lit/ug/slau049f/slau049f.pdf)

[46] *GCC toolchain for the Texas Instruments MSP430 MCUs* (http://mspgcc.sourceforge.net/)

[47] Dunkels A., Gronvall B., Voigt T.: *Contiki - a lightweight and flexible operating system for tiny networked sensors*, Embedded Networked Sensors (EmNets), 2004

[48] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K.: *System architecture directions for networked sensors*, Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[49] Buschmann C., Fischer S.: *Eigenschaften der Funkschnittstelle in Sensornetzen und Auswirkungen für Anwendungen*, 2. GI ITG Fachgespräch Sensornetze, 2004

[50] Lesurf, J.: *The Scots Guide to Electronics* University of St. Andrews, Scotland

[51] Staub T., Bernoulli T., Anwander M., Waelchli M., Braun T.: *Experimental Lifetime Evaluation for MAC Protocols on Real Sensor Hardware*, ACM Workshop on Real-World Wireless Sensor Networks REALWSN, 2006

[52] Ritter H., Schiller J., Voigt T., Dunkels A., Alonso J.: *Experimental evaluation of lifetime bounds for wireless sensor networks*, European Workshop on Wireless Sensor Networks EWSN, 2005