EVALUATION AND CALIBRATION OF SOFTWARE-BASED ENERGY ESTIMATION METHODOLOGIES ON MSB430 SENSOR NODES

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Benjamin Nyffenegger
2010

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

Abstract

This thesis evaluates software-based energy estimation methodologies for wireless sensor nodes in terms of feasibility and estimation accuracy. It contains an extension of todays most widely used estimation model, which incorporates the state transitions of the radio transceiver. The improved model introduces a higher level of granularity which is shown to have a positive impact on the accuracy of the resulting estimations. This claim is backed up with numerous experiments based on the MSB430 sensor node platform with the ScatterWeb [16] operating system and different MAC protocols.
The MAC protocol prototypes used in the tests consist of the CSMA/CA, S-MAC, T-MAC and WiseMAC protocols. The prototype implementations of these protocols are explained in detail in the course of the thesis. To quantify the error rates, we determined the power consumption of a wireless sensor node through hardware-based measurements and estimated the power consumption with an energy estimation model at the same time. The difference between the physically measured energy consumptions and the estimated values served as the statistical measure for the evaluation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The lifetime of a wireless sensor network (WSN) is determined by the energy resources of the individual wireless sensor nodes. Generally, a long lifetime is desired for low maintenance, keeping down monetary costs and last but not least for ecological reasons. This is particularly true for WSNs which are deployed in remote or hostile environments or consist of a large number of nodes. In most cases, the energy resources of the nodes are batteries since energy-scavenging techniques (e.g. harvesting solar energy) can only be employed in very specific scenarios. Sensor nodes with depleted batteries are either recharged, discarded or replaced depending on the environment and the type of WSN nodes used. In hard-to-reach locations, recharging or replacing sensor nodes can be a daunting and cost intensive task. For this reason, there have been many studies in regard to prolonging the lifetime of wireless sensor nodes in recent years.
Along with advancements in energy-efficient hardware, energy-efficient algorithms play an important role in reducing the energy consumption of sensor nodes. Algorithms can be designed for energy efficiency in all levels of the operating system by determining efficient usage patterns of the hardware resources. The efficiency of these algorithms may be validated by determining the resulting energy consumption via hardware-based energy measurements or software-based energy estimation models. Apart from validating the energy efficiency of software, the various energy measurement or estimation methodologies may also be used in energy-aware WSN nodes which base some of their decisions on the current energy consumption. While the accuracy requirements for validating energy-efficient algorithms or energy-aware WSN nodes vary from case to case, it is always beneficial to employ a methodology with a predictable error rate. Software-based energy estimation models have a great appeal for use in wireless sensor nodes because software does not induce considerable additional monetary costs nor does it require additional hardware which may limit the field of application. However, the error rate of these models is not well documented which makes it somewhat unclear whether they can be deployed where accuracy is an issue.
The goal of this thesis is to evaluate software-based energy estimation models in terms of the feasibility and accuracy. The experimental data consists of results from a number of tests with different wireless sensor nodes of the same type and different MAC protocol prototypes. The individual experiments involved physically measuring the power consumption of a node with a Sensor Node Management Device (SNMD) while estimating the consumption of the same node with an energy estimation model at the same time. The mean differences of the physical

measurements to the estimated values served as statistical measures for comparing the different models and quantifying the error rates. In the scope of this thesis, we evaluated two different energy estimation models with four different MAC prototypes on the MSB430 sensor node platform with the ScatterWeb operating system. The first model we examined was the widely spread *three states model* [6] [10]. During the implementation, we discovered a potential weakness of the model as it does not consider the fluctuation of the current draw when switching states. For this reason, we developed the *three states model with state transitions* which models the behavior of the current draw of the state transitions in addition to the previous states of the *three states model*. The different MAC protocols include the energy unconstrained CSMA/CA protocol as it is the most commonly used MAC protocol and the energy-efficient MAC (E^2-MAC) protocols S-MAC, T-MAC and WiseMAC. The reason for choosing S-MAC was because it is the first protocol of its kind and is often included when comparing E^2-MAC protocols. We have chosen the T-MAC protocol because it is the successor of the S-MAC protocol and exhibits different energy consumption characteristics and a shorter end-to-end delay. The S-MAC and the T-MAC protocol were implemented in the scope of this thesis, as they were not readily available for the ScatterWeb operating system at the time. We considered the unsynchronized WiseMAC protocol as a good addition to the other protocols because it introduces a sense of randomness as the individual sensor nodes each wake up at different times.
The structure of this thesis is as follows. Chapter 2 compares the different existing methods for determining the energy consumption and presents the sensor node platform and operating system we used for the experiments. In Chapter 3, we discuss the impact of the MAC protocols on the energy consumption and energy estimation methodologies. Furthermore, we introduce the individual MAC protocols used in the scope of this thesis and provide detailed information about the respective prototype implementations. Chapter 4 contains the experimental evaluation of the energy estimation models with details about the experiment setup, energy estimation models, calibration methodologies and the process of adding sensors to the energy estimation models. The evaluation consists of investigating the power consumption deviations among different sensor nodes and the achieved accuracy of the different models depending on factors such as traffic rate, calibration method and additional sensors. Finally, in Chapter 5 we discuss the results and conclude the thesis.

Chapter 2

Related Work

In this chapter we briefly elaborate on related work focusing on hardware-based energy measurement methodologies and software-based energy estimation methodologies. Therefore, Section 2.1 gives a brief overview of the various measurement and estimation methodologies and compares them in terms of accuracy and applicability.
Also, as part of the related work, we introduce the available hardware and software used within the scope of this thesis. The MSB430 sensor nodes together with the ScatterWeb operating system served as our development platform for all of the energy estimation models and MAC protocol implementations which were used in all the experiments, especially for evaluating the software-based energy estimation models. In Section 2.2 we provide an overview of the MSB430 sensor node platform, and in Section 2.3 we introduce the ScatterWeb operating system.

2.1 Hardware-based Energy Measurements vs. Software-based Energy Estimation

The related work regarding hardware-based energy measurements can be divided into two groups. The first group, discussed in Section 2.1.1, describes the use of oscilloscopes and multimeters to measure the current draw of sensor nodes. The members of the second group, discussed in Section 2.1.2, have developed custom devices to determine the power consumption of the sensor nodes. Section 2.1.3 presents previous work regarding software-based energy estimation methodologies. It includes the work of two research groups that specifically address software-based energy estimation models for sensor nodes. In Section 2.1.4 we recapitulate the pros and cons of the various methodologies.

2.1.1 Physical Measurements Using Oscilloscopes or Multimeters

A number of E^2-MAC articles [15] [3] have described the use of cathode-ray oscilloscopes to measure the power consumption of sensor nodes. The setup, shown in Figure 2.1, requires a device and a low impedance shunt resistor which is connected to a power supply in series. The oscilloscope may then be used to measure and display the voltage drop of the fixed resistor. Because of the linear relationship between the voltage and the current, this representation also

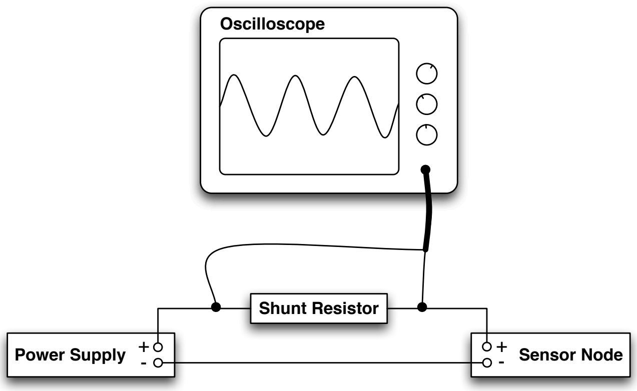

Figure 2.1: Current Measurement Using an Oscilloscope

corresponds to the power consumption of the sensor nodes. This relationship is shown in 2.1 (Ohm's law), where I is the current measured in amperes, V is the voltage measured in volts and R is the resistance measured in ohms. Finally, the continuous signal is sampled in an appropriate resolution and the individual samples stored in a trace file.

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

As an alternative to oscilloscopes, high resolution multimeters may be used to measure the current draw of a sensor node. Typically, a multimeter has the ability to measure the voltage, the current and the resistance with accuracies ranging from 0.5% for standard portable multimeters up to a few parts per million for laboratory grade multimeters. To measure the current draw of a device, the multimeter is connected to the power supply in series so that all current flows through the meter.

The main advantage of both measurement types is that they are considered the *cleanest* methods as they do not involve the addition of any hardware or software components on the sensor nodes. The accuracies of the methodologies come with a few drawbacks, which make them very unpractical to apply it in real world scenarios. The biggest drawback is the complexity of the measurement environment, which requires bulky and expensive hardware that need to be installed on a per-node basis. This leads to experiments bound to controlled laboratory environments with a few sensor nodes being measured rather than real world scenarios in the field with a large number of nodes. Another disadvantage is the size of the trace files, which grow to a large size if the resolution is high. This reduces the possible measurement durations of the experiments instead of being able to conduct long time measurements. For instance, a measurement with a duration of 600 seconds and a sampling frequency of 2000 Hz produces a trace file of approximately 9.6 MBytes (8 digits per sample encoded with UTF-8).

2.1.2 Physical Measurements Using Custom Devices

PowerBench

A methodology to measure the power consumption of the sensor nodes in a testbed was introduced with PowerBench [10]. The testbed features distributed provisions for real-time, hardware-based energy measurements. Each sensor node of the testbed possesses a small interface with power measurement circuitry (shunt resistor and A/D converter) which is connected to an external device that samples the output of the A/D converters. A total of 8 sensor nodes may be connected to a sampling device. The power traces are sent to a central host over Ethernet where they can be analyzed. The advantages of PowerBench are the lower cost and the smaller per-node complexity. The main problem of this approach is that it is not considered a *clean* method to measure the current draw of a sensor node. The argument for this is that hardware or software components that are installed on the sensor nodes to measure power consumption, consume power themselves. Consequently, these components may alter the results of the measurements in contrast to the oscilloscope methodology. Other problems that remain are the size of the power traces and although the per-node complexity is smaller, it is still unfeasible to measure the energy consumption in a large scale deployment of sensor nodes.

SNMD

The Sensor Node Management Device (SNMD) [11] has been developed by the Karlsruhe Institute of Technology to mitigate some of the disadvantages of onboard hardware, oscilloscopes or digital multimeters. The SNMD measures the current and the voltage consumed by a connected sensor node with a specified sampling rate. During a measurement, the measurement results are sent as raw data over a serial interface to a PC where the data is stored in a trace file. The main idea behind the measurement methodology is to control and monitor the energy supply by simulating the battery of the sensor node. It is possible to simulate different battery types and scenarios, e.g. a discharge of the batteries at various temperatures. On the sensor node, the batteries are replaced with a SNMD which controls the voltage supplied to the sensor node and uses USB as its power source. To avoid measurement inaccuracies, the sensor nodes must not draw current from any other source than the power supplied by the SNMD. When measuring power consumption of MSB430 nodes for instance, the nodes may not be connected to a USB interface via the UART interface of the board because the sensor nodes would otherwise draw some current from the UART interface. The SNMD periodically samples the current draw of the sensor node and sends each sample over USB to a connected PC which is used to log the data. Additionally, the SNMD is capable of buffering a number of samples and send them over USB in chunks of 896 kBytes. The sampling resolution is 16 bit and the sampling frequency is configurable up to 20 kHz or 500 kHz when buffering is activated. Controlling the SNMD may be done manually or automated over the USB interface.
The measurement error of the SNMD was rated below 1% [11]. This error was determined using various resistors in the range of 33Ω to 1000Ω. Since resistors have production tolerances of 10%, their values were first measured with high precision laboratory equipment. To take into account the tolerance of the power source, the voltage output of each SNMD was also mea-

sured. Each resistor was connected to a SNMD separately to measure the induced current. The measurement precision of the SNMD results from the comparison of the results measured by the SNMDs and the actual values of the resistors. The first tests showed that the measurement error constantly fluctuated between $300\mu A$ and $400\mu A$. By adding the evaluated error to the measurement results, the measurement error of less than 1% is achieved.
The pros and cons of using this energy measurement methodology is similar to using oscilloscopes or multimeters. Since every node must be connected to a SNMD and every SNMD to a PC, large scale experiments are difficult to apply and require costly hardware even if cheaper than the hardware needed for other hardware-based measurement methodologies. The benefit of this approach is the possibility to accurately measure a small number of devices at a reasonable cost without the negative side-effects of additional onboard sensor node hardware.

2.1.3 Software-based Energy Estimation Models

Software-based On-line Energy Estimation for Sensor Nodes

[6] has proposed a software-based, on-line energy estimation mechanism to simplify the process of the power consumption evaluation and allow intelligent decisions of sensor node protocols based on the energy usage. The article describes a software model that is simple to integrate into existing sensor node operating systems. Furthermore, the article evaluates the accuracy of the mechanism by comparing the results to the lifetime of the sensor nodes. The software estimation mechanism stores a timestamp every time a component of a sensor node is turned on or off. The software needed for logging is added to the drivers of the components and has a small memory footprint. The mathematical model used to derive the total power consumption E is shown in the Equation 2.2, where V is the supply voltage, I_m the current draw of the microprocessor when running, t_m the time in which the microprocessor has been running, I_l the current draw of the microprocessor in low power mode, t_l the time in which the microprocessor spent in lower power mode, I_t the current draw of the device in transmit mode, t_t the time which was spent transmitting, I_r the current draw of the device when receiving, t_r the time spent receiving, I_{c_i} the current draw of other components such as sensors or LEDs and t_{c_i} the time which was spent using the other components of the device.

$$\frac{E}{V} = I_m t_m + I_l t_l + I_t t_t + I_r t_r + \sum_i I_{c_i} t_{c_i} \tag{2.2}$$

The values for I_m, I_l, I_t, I_r and I_{c_i} were acquired with averaging off-line oscilloscope measurements of the current draws of the respective sensor node states. For evaluation, the sensor nodes were equipped with capacitors which have a predictable energy storage. First, the power supplies of a sensor node were turned on to fully charge the capacitor. When the power supply was turned off, the sensor nodes ran on the energy stored in the capacitors and lasted for a few minutes depending on the program running. A program consisted of turning components on and off at regular intervals. During such a test, the nodes transmitted their energy estimation to a base station node which was connected to a PC for logging purposes. The accuracy of the energy estimations was determined by comparing the estimations with the capacity of the capacitor. The tests of the article were performed with a network of nine sensor nodes running the

X-MAC protocol [3], which turns the radio on and off in regular intervals to reduce the power consumption. The article does not provide any results in form of error rates, instead it states that *further study is needed to accurately quantify the error rate of the mechanism.* This is partly due to the difficult evaluation mechanism, which had a large margin for error as it involved the manual turning on and off of the nodes to charge and discharge the capacitors. The article recommends the validation of the software-based energy estimation mechanism with hardware-based measurements for future work.

PowerBench

The research group behind PowerBench [10] has also implemented a software-based energy estimation mechanism on the sensor nodes of their testbed and validated the results with their hardware-based energy measurements. The estimation model used by PowerBench is similar to the model used by [6]. The equation is shown in 2.3, where E is the energy consumption, V the supply voltage, i the index of the sensor nodes state, I the current draw of state i and t the time spent in state i.

$$\frac{E}{V} = \sum_i I_i t_i \tag{2.3}$$

Experiments were made using a *three states model* with the states *receive*, *transmit* and *sleep* to estimate the power consumption of the individual nodes in the testbed. B-MAC [15] and Crankshaft [9] were used as MAC protocols. The current draw of each sensor node state was determined in the same way as in [6], through averaging the current draw values of each state, which were acquired through hardware-based measurements. The average current draw values were calculated from the measurement results across all measured sensor nodes and additionally, from the measurement results of each node individually (per-node calibration). While the errors of individual runs for the tests with average current draw values, which were calculated across all sensor nodes, reached up to 21%, the estimates with per-node calibration were far more accurate. On average, the estimates for B-MAC were off by 2% and the estimates for Crankshaft were off by 13%. The article claims that the worse estimates for Crankshaft are a result of an inefficient implementation of the Crankshaft protocol. Crankshaft is a slotted protocol, and the particular implementation showed frequent, small current draw outbursts, which were induced by the CPU in every slot. It argues that these current draw outbursts are responsible for the inaccuracies, as they are not modeled in the estimation model used for the tests. The conclusion is that the *three states model* is an accurate software-based energy estimation model provided that the CPU is not frequently used. It is also mentioned in the beginning of the article, that using additional radio state transitions might result in better estimation results. No tests were made however, using such state transitions.

2.1.4 Hardware-based and Software-based Methodologies Compared

All of the described methodologies to determine the energy consumption of sensor nodes have their particular pros and cons. However, it is possible to compare the hardware-based methodologies (oscilloscopes, multimeters, custom devices) to the software-based estimation methodologies as the methodologies of each group share the same basic characteristics. Table 2.1

is a compilation of the pros and cons as mentioned above in the description of the respective methodologies. The main differences of the two groups can be boiled down to the degrees of accuracy and complexity. While hardware-based methods inevitably feature a higher accuracy than software-based estimations, they require expensive equipment which needs to be set up on a per-node basis. This leads to experiments bound to a laboratory environment and limited to a small number of sensor nodes. In contrast, the deployment of a software-based mechanism is not limited by the cost, time expenditure or any physical constraint through the size of the equipment.

Table 2.1: Hardware-based energy measurements vs. software-based energy estimations

Hardware-based Energy Measurements	Software-based Energy Estimations
+ High accuracy. Depends on the accuracy of the measurement hardware.	- Less accurate than the hardware-based. Depends on implementation of the software and the mathematical model.
+ Does not require calibration.	- Calibration of the parameters can be very time-consuming.
- Large trace files.	+ Small memory footprint.
- Setup of measurement equipment may be time consuming.	+ Quick set up as there is no special equipment required.
- Requires costly hardware on a per node basis.	+ Only requires hardware-based energy measurement devices for calibration.
- More or less bound to laboratory tests.	+ Suitable for field tests.
- Limited to a small number of sensor nodes.	+ May be applied on large deployments of sensor nodes.

2.2 MSB430 Sensor Node Platform

The Modular Sensor Board (MSB) [2] is a research platform distributed by the ScatterWeb GmbH [16]. It features a MSP430 series microcontroller (MCU), a wireless radio chip, external storage and two sensors. The MSP430F1612 MCU is clocked between 100 kHz and 11 MHz by a software configurable digital oscillator (DCO) and provides 5 KB of RAM and 55 KB of Flash-ROM. For external storage, a Secure Digital (SD) card slot may be used to store up to 32 GB with Secure Digital High-Capacity (SDHC) cards. The radio on board is a Chipcon CC1020 transceiver with a low-noise amplifier on the receiver. It uses the license-free 868 MHz ISM band (863 - 870 MHz) and has an output power of up to 8.6 dBm (7.2 mW). The band is divided into 8 channels with a data rate of 19.2 kbit/s each when using Manchester encoding. Both, the channel as well as the desired output power of the transceiver can be selected via software. One of the two sensors is the Sensirion SHT11 [17], for measuring the temperature and relative humidity. The second sensor on board is the MMA7260Q [8], an accelerometer from Freescale, which measures acceleration in three axes to allow movement detection. For input and output there are 18 digital pins available and an analog-to-digital converter (ADC) as well as a digital-to-analog converter (DAC). Furthermore, a JTAG and an UART interface are available for programming and debugging. The sensor node may be connected to the USB interface of the PC through the UART interface with a FTDI converter-cable. Power for the board may come from three AAA batteries or an USB interface. Switching from one power source to the other can be done with a hardware switch.

2.3 ScatterWeb Operating System

The ScatterWeb [16] operating system has been developed by Freie Universität Berlin and ported from the prior ScatterWeb software running on the so-called ESB sensor nodes to support the MSB430 sensor nodes. This relatively simple operating system is event-driven rather than multi-threaded. In a multi-threaded system, blocked threads are waiting for events and unblocked in the occurrence of events. Each thread requires a stack and runs until the next blocking statement. This leads to a large memory usage which is not feasible on the MSB430 sensor nodes. In an event-driven system like ScatterWeb, the kernel calls an event handler in case of an event. These event handlers cannot be preempted by other event handlers and run until completion. Due to this, the memory usage is smaller, but event handlers with long computations may block the system for a long time. It is particularly dangerous to add time consuming computations to the interrupt routines of the transceiver. This may have negative effects on certain MAC protocols as they must be able to switch states very quickly.
The mentioned kernel of the ScatterWeb operating system is not like a traditional kernel used in desktop computer operating systems in a sense that it does not nest in its own memory space. It shares the memory space with the applications and basically consists of a simple *superloop* function, which runs for the lifetime of the node, discovers events and calls the appropriate handlers. In the case of a transmission for instance, the packet is assembled, stored in a buffer and a flag set as an indicator for a packet ready to send. On the next iteration of the *superloop*, this flag is recognized and the appropriate action executed.

Chapter 3

Implementation of MAC Protocols

In this chapter we describe the fundamentals of MAC protocols in wireless sensor networks (Section 3.1) and discuss in what ways MAC protocols have an impact on the energy consumption and the accuracy of energy estimation mechanisms of wireless sensor nodes.
The next sections in this chapter describe the four MAC protocols that were used to evaluate the accuracy of the energy estimation methodologies implemented in this thesis. One of the MAC protocols is the energy unconstrained CSMA/CA protocol (Section 3.2), and the remaining three are the energy conserving protocols (E^2-MAC) S-MAC (Section 3.3), T-MAC (Section 3.4), and WiseMAC (Section 3.5). At the end of every section, we describe a typical, real world protocol scenario with a graphical representation, acquired through power measurements. For every MAC protocol, a scenario was chosen which highlights its unique properties.
In the scope of this thesis, S-MAC and T-MAC were implemented on the MSB430 platform. They are referred to as synchronized protocols and perform clock synchronization through the periodic exchange of synchronization packets. Section 3.3.1 discusses the methods used for clock synchronization and the achieved accuracy. CSMA/CA is implemented as the default MAC protocol of the ScatterWeb operating system and the implementation of the WiseMAC protocol was provided by P. Hurni, University of Berne [12]. The following sections explain the basic modes of operation of the various protocols as well as their implementation specific parameters and packet formats.

3.1 Impact of MAC Protocols on the Energy Consumption and Energy Estimation Models of Wireless Sensor Nodes

Generally speaking, a Medium Access Control (MAC) protocol in a wireless sensor network (WSN) defines the rules how a shared physical medium is accessed and how other nodes are addressed. Additionally, the protocols may be responsible of performing error detection and correction as well as the optional retransmission of packets in case of uncorrectable errors or missing acknowledgements. The members of the network are usually attributed a unique address to identify the sender and receiver. The physical medium is usually a certain spectrum of the radio frequency which is often divided into multiple channels. The transceiver of a wireless device cannot transmit and receive data on one channel at the same time because of interfer-

ences that render the signal useless. To achieve communication in both ways, time multiplexing with alternating transmitting and receiving or frequency multiplexing with separate channels for transmitting and receiving may be used. The latter requires multiple radios, which is undesirable as it increases the cost, power usage and complexity of the sensor nodes. Since the medium is also shared by a number of network nodes, a MAC protocol may implement ways to avoid, prevent and handle collisions. The effectiveness of how the MAC protocol handles these challenges has a great impact on the energy consumption of a wireless sensor node. The reason for this is that the transceiver of a wireless sensor node is often responsible for consuming the most part of the total energy consumption. In the case of the MSB430 sensor nodes for instance, a device that is in an idle listening state consumes roughly 91% more power than it does when it has turned off the transceiver. This ratio rises to 96% when the transceiver is transmitting data with the highest transmission power possible. Since MAC protocols dictate the usage patterns of the devices' transceivers, they are also responsible for the efficient usage of this resource. Clearly, Energy-Efficient MAC (E^2-MAC) protocols have a large potential to extend the lifetime of sensor nodes when they are running on a limited energy source. This is also reflected in the large amount of studies devoted to E^2-MAC protocols in recent years.
Not only does the choice of the E^2-MAC protocol have an impact on the energy consumption of the wireless sensor nodes but also on the accuracy of the software models that estimate the energy consumption. This is mainly due to small errors in the predetermination of the current draw of the sensor node's states and the simplicity of the energy estimation models. Each E^2-MAC protocol produces a different usage pattern of the states, e.g. the time spent idle listening varies from protocol to protocol. The impact of one of these errors may vary depending on the usage pattern of sensor node states, e.g. an error in the current draw value for idle listening has a greater impact on the estimated energy consumption of E^2-MAC protocols that spend more time idle listening than others.

3.2 CSMA/CA

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) [4] is the most commonly used (IEEE 802.11 [13]) energy unconstrained MAC protocol which senses a shared medium before transmitting data. It is energy unconstrained because the protocol does not implement any mechanism to save energy whatsoever. It can be seen as the worst case scenario in terms of energy consumption for a single channel MAC protocol. For this reason, it serves as a good basis for comparing MAC protocols that have the objective to reduce the power consumption. Carrier sensing refers to the process of probing a medium, which can be an electrical conductor or a certain frequency band, for a signal.

3.2.1 CSMA/CA Protocol Operation

The basic scheme of the protocol is as follows. The source node generates a random backoff time before attempting to send a packet. Afterwards, the source stands by until the backoff time has run out and samples the medium again to detect a potential ongoing transmission. If a signal is found on the medium, the sender waits for another random backoff time and probes

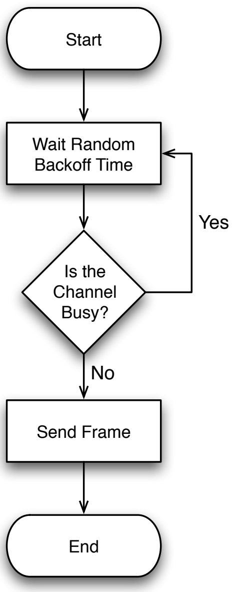

Figure 3.1: CSMA/CA flowchart

the medium once more. Finally, if the sender doesn't sense a signal, it may send the frame. This process is referred to as collision avoidance. Figure 3.1 shows the flowchart of the CSMA/CA protocol. How many times a source waits for a random backoff time, when it has encountered a busy medium, is defined by the *number of retries*. When the *number of retries* is exceeded the packet will be dropped. With collision avoidance, collisions can be largely avoided but not entirely eliminated as the *maximum random backoff time* is usually rather short for efficiency reasons. Low values for the backoff times result in shorter packet delays as the packets get sent earlier. Also, the lower the limit of the backoff time, the higher the probability of multiple nodes simultaneously generating the same backoff time and thus sending a packet a the same time.

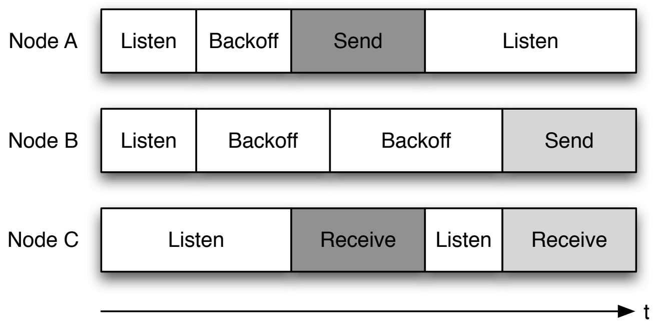

Figure 3.2: Multiple nodes intend to send simultaneously with CSMA/CA

Figure 3.2 shows a CSMA/CA scenario with three nodes A, B and C. The starting situation in the figure is that nodes A and B have a frame in the transmit buffer at the same time. It continues with node A generating a shorter backoff time than B and is thus able to send the frame after it has sensed that the medium is free of communication. After the backoff time of node B has run out, node B senses the ongoing communication of node A and generates another random backoff time ($\text{number of retries} > 0$). In the end node B can send its frame after it has sensed that the medium is not busy anymore.

3.2.2 MSB430 CSMA/CA Prototype Packet Specification

Table 3.1 shows the various fields and their corresponding sizes of a CSMA packet used for the implementation. The preamble contains a bit pattern used to synchronize the sender's and the receiver's transceiver and the subsequent start delimiter denotes the beginning of the actual data. Currently, the ScatterWeb implementation allows for 1 byte node identifiers. Consequently, the fields *address target* and *address source* allow for addresses in the range of 0-255. The addresses between 0-254 are used for addressing individual sensor nodes and the address 255 is used for broadcasting packets. The field *flags* can be used for either deactivating Cyclic Redundancy Check (CRC) error checking or to specifically request acknowledgements for data packets. For all the experiments in this thesis, CRC error checking was activated and ACK packets were always requested, however, only for acknowledging data packets. Source nodes number their packets sequentially and store the number of the packets in the field *number*. This helps receivers to identify retransmitted packets and drop them if the packet has already been processed. The *type* field is used to evaluate what kind of packet the receiver is processing and may therefore decide to forward it to the application layer or deal with it directly on the MAC layer. The 16-bit CRC (CRC-16) checksum is calculated over the entire frame and appended at the end of the packet.

Table 3.1: CSMA packet format

Field	Bytes	Description
Preamble	6	Bit sequence for bit synchronization of transceiver
Start Delimiter	2	Indicates the beginning of the data
Size	1	The size of the packet including payload
Address Target	1	The address of the receiver (0 - 255)
Flags	1	Deactivation of CRC error checking or ACK requests
Address Source	1	The address of the sender (0 - 255)
Number	1	Packets are numbered sequentially
Type	1	User defined packet types, ACK packet, etc.
Payload	variable	The data to be sent
CRC	2	The CRC checksum

3.2.3 MSB430 CSMA/CA Prototype Parameters

There are three main parameters in the default ScatterWeb-based CSMA/CA implementation on the MSB430 nodes. They can be adjusted to find a good balance between packet delay and the probability of a collision. All the parameters were determined experimentally in a multihop environment with various traffic. The parameter for *maximum random backoff time* was set to 23 ms and the parameter for the *number of retries* was set to 1. Furthermore, the *number of retransmissions* was set to 1. The last value stands for the number of packet retransmissions in case of missing acknowledgments. High values for *number of retries* and *number of retransmissions* could result in a buffer overflow if there are many collisions in scenarios with high network traffic. Since the buffer size of the MSB430 nodes is very small, the values for these parameters are preferably kept low. However, the desired network traffic in the experiments of this thesis can easily be sustained with these parameter values without any packet loss.

3.2.4 Typical MSB430 CSMA/CA Prototype Current Trace

Figure 3.3 is a graphical representation of a sensor node forwarding a packet with CSMA/CA. The data for this figure was acquired with a SNMD power measurement (see Chapter 4) of a MSB430 sensor node with the transmission power set to the maximum value. The graph shown in the figure is the current draw I of the sensor node at time t.

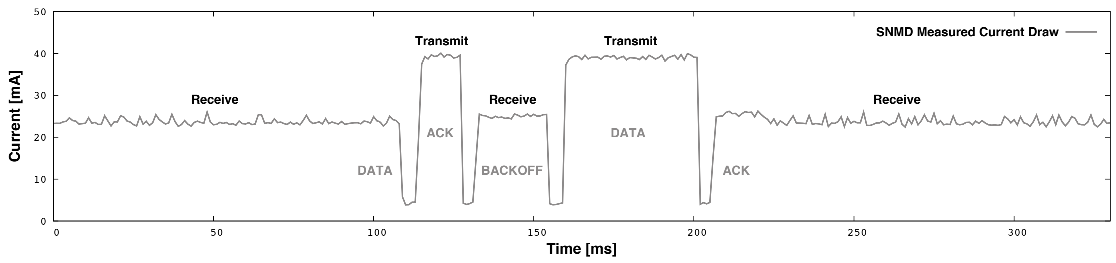

Figure 3.3: Current Draw of a Sensor Node Forwarding a Packet with CSMA/CA

The sensor node receives a packet at $t \approx 95$ ms. It immediately acknowledges the packet, generates a random backoff time and listens for traffic on the medium while waiting for the backoff time to pass. The node has not sensed any traffic during the backoff time and transmits the packet at $t \approx 160$ ms. Shortly afterwards, it receives the ACK for its previous transmission and keeps the radio turned on, ready to receive data.

3.3 S-MAC

The Sensor-MAC (S-MAC) [19] protocol is specifically designed for sensor networks, and it is known as the first protocol with the intention to reduce the energy consumption of a wireless sensor node. The reduction of energy consumption is accomplished by minimizing overhearing, control packet overhead, idle listening and avoiding collisions. Overhearing is the reception of data packets intended for other nodes. Collisions are a result of simultaneous transmissions or other interferences. Control packets are necessary for controlling the proper flow of a MAC protocol. However, it is desired to prevent the transmission of unnecessary control packets in order to save energy. Idle listening is the process of listening to a channel and waiting for potential traffic.

3.3.1 S-MAC Protocol Operation

S-MAC minimizes the sources of energy waste at the cost of throughput and latency by letting the nodes go to sleep after a short listening period. Figure 3.4 shows the interval in which the nodes periodically wake-up and go to sleep to shorten the idle listening time. The duration of the interval and ratio between listen and sleep time affect the energy saving capability, throughput and latency. Thus, the values for the duration and ratio parameters are adjusted depending on the field of application. The listen phase of the interval is further divided into two phases, the SYNC phase is for clock synchronization purposes and the RTS/CTS phase is to advertise, send and receive data. These phases will be discussed in the following.

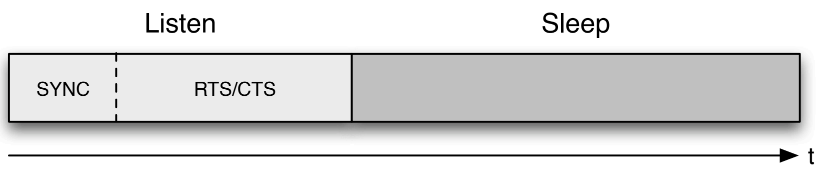

Figure 3.4: S-MAC states

Synchronization Phase

Sensor nodes need to wake-up and go to sleep in a synchronized fashion. For this purpose, the sensor nodes perform an initial time synchronization and periodically synchronize their clocks in order to compensate for the clock drift of the nodes. This time synchronization is important for the S-MAC protocol, however the accuracy is not as crucial as in TDMA.
Each sensor node in a S-MAC network maintains a schedule with a certain wake-up time which designates the beginning of the SYNC phase. In order to communicate, the sensor nodes must adapt the schedules of their direct neighbors. This schedule adaption is executed by each sensor node upon entering a network. First, the node listens for SYNC packets, which contain the schedules, for a certain amount of time. If the node does not receive a SYNC packet during that time, it begins its wake-up and sleep cycle and broadcasts the SYNC packet as explained at the end of this section. If the node receives a SYNC packet during the initial listening time, it sets its clock accordingly and rebroadcasts the SYNC packet with the schedule.
In a multi-hop network, groups of neighboring sensor nodes may form clusters that maintain different schedules. In this case, sensor nodes that are located in the transmission range of multiple clusters may form a bridge for those clusters by maintaining all the different schedules. Maintaining different schedules means that the sensor node wakes up on each wake-up time of every schedule. Multiple schedules may be adapted by a sensor node if it receives multiple SYNC packets with different schedules during the initial listening time. However, the scheme for adapting multiple schedules was not implemented as part of this thesis as the experiment setups did not incorporate sensor node clusters.

Unidirectional Synchronization

For clock synchronization, a unidirectional synchronization scheme was chosen to keep the protocol overhead at a minimum. The unidirectional synchronization process requires the transmission of just one packet with a timestamp. Figure 3.5 shows such a synchronization scenario with a node A sending a synchronization packet to a node B. Node B receives time T_1 from node A at time T_2. To set the clock node B has to know the *packet delay* T_D in order to compute $T_2 = T_1 + T_D$.

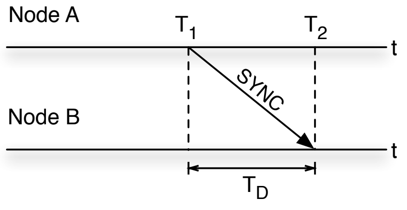

Figure 3.5: Unidirectional synchronization

The *packet delay* T_D consists of the *send time*, *access time*, *transmission time*, *propagation time*, *reception time* and *receive time*. The *send time* is the time needed by the application layer to pass the packet on to the MAC layer and for the MAC layer to assemble a packet ready for transmission. This time depends on software delays and on the size of the packet. In the case of SYNC packets, the time for passing it on to the MAC layer can be neglected because the SYNC packets are assembled directly on the MAC layer. The time for assembly is constant because of the constant size of the SYNC packet. The *access time* is the time it takes until the medium can be accessed. This value is variable and dependent on the MAC protocol. It was necessary to avoid this variability and gain a constant *packet delay* for the unidirectional synchronization scheme used by S-MAC and T-MAC. For this reason, the sensor node must insert the current time T_1 into the SYNC packet after it has already been assembled and after any delay caused by medium access (contention with random backoff time). This is done by inserting the time data into the SYNC packet directly in the transmission interrupt service routine, which designates the last possibility to alter the contents of a packet before it is sent by the radio byte-by-byte. The *transmission time* is the time it takes for a radio to transmit a packet byte-by-byte. It is dependent on packet size and radio speed. The *transmission time* is again of a fixed duration since the size of the SYNC packets is constant. The *propagation time* is the time it takes for the data to travel from the source to the receiver. This time is negligible since the propagation speed of electromagnetic waves is $3.00 * 10^8 m/s$, and the distance of the nodes to each other is usually too small to have an impact on the *packet delay*. The *reception time* is the time required to receive all the bits of the packet. This is also constant, just like the *transmission time*, as the size of the SYNC packets do not vary. The last component of the *packet delay* is the *receive time*, which is the time for disassembling the packet on the MAC layer and handing it to the application layer. Like the *send time*, this value is variable and dependent on packet size and software delays. The SYNC packet is not handed over to the application layer and the time for disassembly is constant because of the constant SYNC size.

To distribute these SYNC packets, every node tries to broadcast such a packet in the SYNC phase of the S-MAC interval. To prevent collisions and unnecessary transmissions of packets, the nodes implement a simple collision avoidance mechanism just like the one found in CSMA/CA. At the beginning of the SYNC phase, the nodes wait for a random backoff time before broadcasting the SYNC packets. If a node receives a SYNC packet while waiting, it aborts the transmission of the SYNC packet and sets its time according to the received information in the packet. To minimize unnecessary traffic, the SYNC packets are not sent in every cycle. Instead, SYNC packets are sent at a chosen interval which is small enough to prevent the clocks from drifting apart too much and prevent long waiting times for new sensor nodes entering the network. The resulting accuracy of this synchronization scheme is shown in the following.

Packet Delay and Synchronization Accuracy Measurements

The *packet delay* for the SYNC packets had to be determined empirically for the actual implementation of the S-MAC synchronization scheme. The reason for this was that while some of the factors accounting for the *packet delay*, like the *transmission time*, are simple to predict merely with the knowledge of the hardware and packet specifications, however others, like the *send time*, are rather difficult to predict as they are dependent on many factors of the protocol implementation and hardware. The basic idea to determine the *packet delay* was to compare the clocks of two sensor nodes after synchronization while neglecting the *packet delay* T_D. The resulting difference of the clocks would represent the unknown value for T_D.
The hardware setup for the tests consisted of two MSB430 sensor nodes connected to a PC via the serial interface. Both of the sensor nodes were programmed to print a character to the console output of the PC at the beginning of each second. One of the sensor nodes had the additional task of broadcasting a SYNC packet in a fixed interval. The sensor node receiving the SYNC packets set its clock to $T_2 = T_1 + T_D$, as illustrated in Figure 3.5. When the *packet delay* T_D is set to 0 ms, the mean difference of the sensor nodes' clocks after synchronization represents an approximation of the *packet delay* T_D. The accuracy of the synchronization can then be evaluated by setting the appropriate value for T_D. For each of the sensor nodes, a program on the PC stored a log entry with a timestamp each time a node printed a character to the console. The logging task on the PC has an inaccuracy of 1-2 ms due to the scheduler of the Linux kernel (with CONFIG_HZ set to 1000, the default value for Ubuntu). The Tables 3.2 and 3.3 each contain a 4 second excerpt of such a log file. The character # on the beginning of each row is the character that was printed to the console by the sensor node. This character is followed by the timestamp, which has a resolution of microseconds. Finally, the time difference of two timestamps in the respective row of each log file represents the difference of the clocks C_{diff} at

Table 3.2: Log file Node A

Node A
2009-10-20 18:09.03 324577
2009-10-20 18:09.04 325553
2009-10-20 18:09.05 326512
2009-10-20 18:09.06 325500

Table 3.3: Log file Node B

Node B
2009-10-20 18:09.03 324659
2009-10-20 18:09.04 324569
2009-10-20 18:09.05 324521
2009-10-20 18:09.06 325679

any given time *t*.
Figure 3.6 shows the results of the first experiment with the synchronization interval set to 10 sec and T_D set to 0 ms. In the beginning of the test, the clock difference of the two sensor nodes stood at an average of 307 ms. After the reception of the first SYNC packet at $t = 63$ sec, the clock difference dropped to an average of 17.7 ms. In the second test, the SYNC interval was also set to 10 sec. The value for T_D, however, was set to 18 ms to test for the resulting accuracy of the approximated *packet delay*. The result of the second experiment is shown in Figure 3.6 with an average clock difference of 1 ms after synchronizing. This experiment scheme was repeated with a SYNC interval of 60 sec and a T_D of 0 ms. The experiment resulted in an average clock difference of 18.2 ms clock after synchronization as seen in Figure 3.8. The last

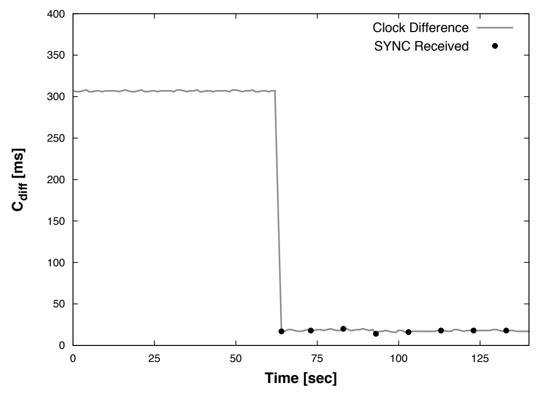

Figure 3.6: SYNC interval: 10 sec, *packet delay* T_D: 0 ms, mean(C_{diff} after SYNC): 17.7 ms median(C_{diff} after SYNC): 18 ms

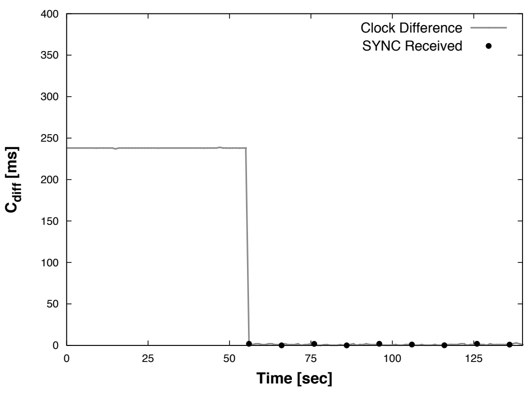

Figure 3.7: SYNC interval: 10 sec, *packet delay* T_D: 18 ms, mean(C_{diff} after SYNC): 1.0 ms median(C_{diff} after SYNC): 1 ms

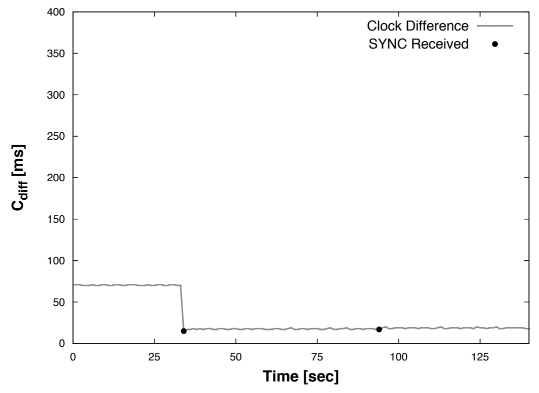

Figure 3.8: SYNC interval: 60 sec, *packet delay* T_D: 0 ms, mean (C_{diff} after SYNC): 18.2 ms, median(C_{diff} after SYNC): 18 ms

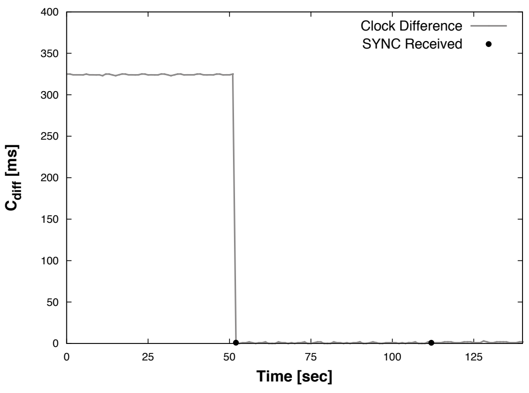

Figure 3.9: SYNC interval: 60 sec, *packet delay* T_D: 18 ms, mean(C_{diff} after SYNC): 1.3 ms, median(C_{diff} after SYNC): 1 ms

experiment with the SYNC interval set to 18 ms and T_D set to the approximated *packet delay* value of 18 ms, shown in Figure 3.9, again resulted in a mean clock difference of 1 ms. From the results of these experiments, we infer an accuracy of $\sim$1 ms for the S-MAC synchronization.

RTS/CTS Phase

The SYNC phase of the S-MAC protocol is followed by the ready-to-send/clear-to-send (RTS/CTS) phase in which the nodes advertise data to other nodes and, if successful, start with the transmission. Figure 3.10 shows a flowchart of the RTS/CTS phase. A node that wishes to transmit data sends an RTS frame to the destination node while performing collision avoidance through carrier sensing, identical to the scheme used in the SYNC phase. The RTS packet

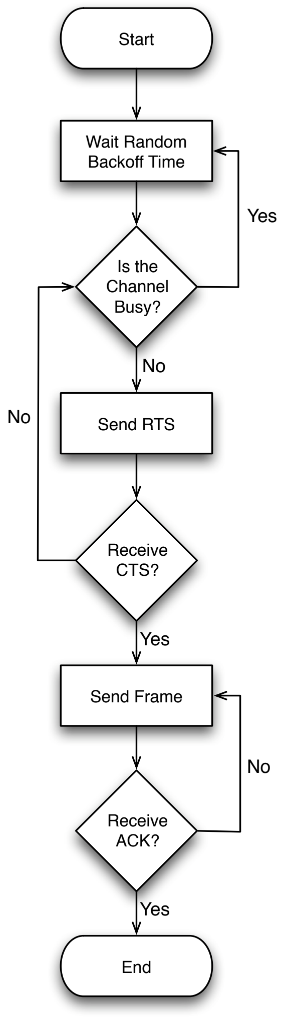

Figure 3.10: RTS/CTS flowchart

contains information about the number of packets that the sender intends to transmit to the receiver. This number is stored in all nodes that receive the RTS frame. It is called the network allocation vector (NAV). By consulting the NAV, the other nodes in the network know if the network is currently busy or not, and the receiver knows how many packets it can expect from the sender. Upon reception of a RTS frame, the destination node replies with a CTS frame without performing any collision avoidance. Only after the reception of this CTS frame, the sender may begin to transmit the packets. Each packet has to be acknowledged with an ACK from the receiver, otherwise it will get retransmitted by the sender after a predefined time-out. This transmission process may go on until the end of the interval (listen/sleep period). If a node has more packets to send in the buffer, it has to repeat the process in the next listen period after the SYNC phase has finished. If the conversation is over (the source has no more packets to send) before the interval is over, the nodes go to sleep until the next scheduled wake-up. By checking the current NAV, the receiver knows if the sender has finished sending packets or not, and can thus go to sleep immediately after sending the ACK packet for the last data packet of the sender. A node overhearing a RTS/CTS conversation immediately turns off its radio (goes to sleep) and wakes up in the next interval for the SYNC phase. This way, other nodes do not interfere with nor overhear the conversations of other nodes. Reducing the number of interferences and the time spent overhearing leads to less collisions and less wasted energy.
Fairness issues are addressed with the computation of the random backoff time before sending an RTS packet. Every node with the intention to send packets gets a fair chance in the beginning of the RTS/CTS phase by computing a random backoff time. The node with the shortest backoff time wins the medium. Fairness is guaranteed, as all sensor nodes of a WSN have the same chance of generating the shortest backoff time (the expected value of the backoff time is identical for all sensor nodes).

3.3.2 MSB430 S-MAC Prototype Packet Specification

Table 3.4 shows the fields and the coresponding sizes of a S-MAC packet. The only difference to the format used by the CSMA protocol is the *Timestamp / NAV* field. The timestamp value is 11 Bits long and represents the current milliseconds of the node. This length is enough for

Table 3.4: S-MAC packet format

Field	Bytes	Description
Preamble	6	Bit sequence for bit synchronization of transceiver
Start Delimiter	2	Indicates the beginning of the data
Size	1	The size of the packet including payload
Address Target	1	The address of the receiver (0 - 255)
Flags	1	Deactivation of CRC error checking or ACK requests
Address Source	1	The address of the sender (0 - 255)
Number	1	Packets are numbered sequentially
Type	1	ACK packet, RTS packet, SYNC packet, etc.
Timestamp / NAV	**2**	**Time synchronization and network allocation vector**
Payload	variable	The data to be sent
CRC	2	The CRC checksum

the milliseconds information as the values range from 0-999. S-MAC does not require absolute synchronization of the clock (hrs:sec:ms). For the implementation at hand, it is sufficient for the nodes to synchronize only their current milliseconds to begin the listen/sleep cycle at the same time. The value for the network allocation vector (NAV) is 5 bits long and contains the number of packets that the sender is buffering. In S-MAC only receivers make use of this figure for knowing when the last packet from the sender has been transmitted.

3.3.3 MSB430 S-MAC Prototype Parameters

The *listen/sleep cycle* defines the duration of one full interval as shown in Figure 3.4. We have set it to 1000 ms. The duration of the *SYNC* phase was set to 50 ms and the RTS/CTS phase was set to 100 ms. The values were first determined in consideration with the theoretical hardware and software constraints and then adjusted by experimenting with multiple nodes and traffic rates to achieve robustness with little collisions and retransmissions. The hardware constraints *access time*, *transmission time* and *reception time* are embodied in the value *packet delay*. The duration of the *packet delay* was investigated empirically and is explained in Section 3.3.1. The software constraint is the maximum possible random backoff time (contention). This figure is also somewhat bound to the given hardware as it is not an absolute time declaration but merely a counter that is decremented in an interrupt. The maximum random backoff time was set to 32 ms for SYNC packets and 39 ms for RTS packets. Experiments showed evidence that these values provide enough headroom to reliably prevent collisions. The following equation shows how the duration of the *SYNC* phase is derived.

$$\begin{aligned} \text{SYNC} &= max(\text{backoff time}) + \text{packet delay} \\ &= 32\text{ms} + 18\text{ms} = 50\text{ms} \end{aligned}$$

A short SYNC phase is desired to reduce energy waste as S-MAC declares this phase as mandatory. However, the nodes do not perform synchronization in each cycle. To minimize the duration of the SYNC phase, a lower value was chosen for the random backoff time and the resulting 50 ms was not adjusted with any headroom to provide stability. A lower maximum backoff time results in more collisions as the probability of multiple nodes generating similar backoff times, and thus creating interferences, increases. This could result in an extreme situation where all SYNC packets collide and none of the nodes are able to synchronize their clocks. However, with a setup of only three sensor nodes, as used in the experiments for this thesis, a situation like this is rare. Additionally, nodes which do not receive nor send a SYNC packet when the SYNC interval calls for it just defer synchronization to the next interval.
Some of the ideas behind the duration of the SYNC phase also apply to the duration of the RTS/CTS phase. However, there are some important differences. One of those is the higher maximum backoff time. As mentioned before, higher maximum backoff times result in less collisions, and colliding RTS packets have a greater negative impact on the energy consumption as opposed to SYNC packet collisions. Thus, to provide less chances for multiple nodes to generate similar backoff times, we have raised the value to 39 ms. Another difference to the SYNC phase is that the RTS/CTS phase must also provide enough time for the receiver to return a CTS packet. For this reason, the time it takes to receive the CTS frame (*packet delay*) has to be

added to the formula. Since the CTS packet is sent by the receiver without contention, there is no need to add the backoff time. Finally, an adjustment value of 25 ms was added as the service quality requirements in this phase are different than in the SYNC phase. The adjustment value allows some headroom for retransmissions of RTS/CTS packets in case of errors and enhances the overall stability of the protocol. The equation below shows the derivation of the duration of the *RTS/CTS* phase.

$$\begin{aligned}\mathrm{RTS/CTS} &= max(\text{backoff time}) + 2(\text{packet delay}) + \text{adjustment value}\\ &= 39\text{ms} + 36\text{ms} + 25\text{ms} = 100\text{ms}\end{aligned}$$

Allthough the clock drift of the nodes would allow for much larger synchronization intervals (see Section 3.3.1), the chosen interval is much smaller. Performing synchronization every five cycles improved the overall robustness of the protocol as the duration of the SYNC phase is very short and does not have any service guarantees.

3.3.4 Typical MSB430 S-MAC Prototype Current Trace

Figure 3.11 is a graphical representation of a sensor node sending a packet with S-MAC. The data for this figure was acquired with a SNMD power measurement (see Chapter 4) of a MSB430 sensor node with the transmission power set to the maximum value. The graph shown in the figure is the current draw I of the sensor node at time t.

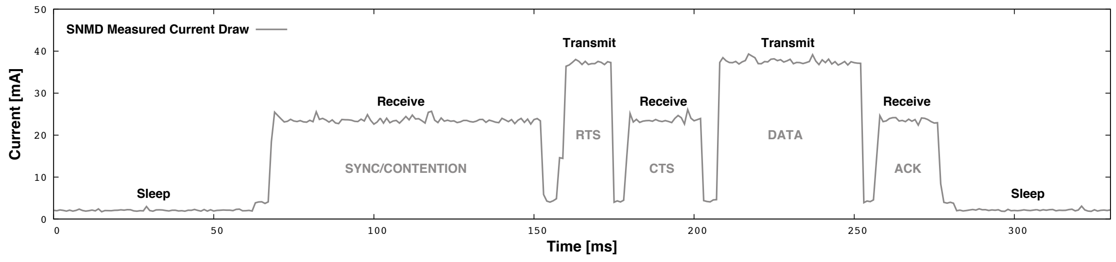

Figure 3.11: Current Draw of a Sensor Node Sending a Packet with S-MAC

The sequence is just long enough to show the current draw of every possible state of the onboard radio of the sensor node. The sensor node wakes up from the sleep state at $t = 69$ ms and turns on the radio to listen for potential synchronization packets. Whether the sensor node has actually received a SYNC packet during the SYNC phase can not be established purely by analyzing the trace file. This is because in most nodes there is no conceivable difference between receiving a packet and idle listening in terms of the current draw. The SYNC phase is followed by the the RTS/CTS phase 50 ms later, at $t = 119$ ms. The sensor node has generated a random backoff time of about 34 ms and switches to the transmit state at $t = 153$ ms to send a RTS packet. Apparently, it generated the shortest backoff time or there was no other node that contended for the medium. Directly after sending the RTS packet, the node switches to the receive state and receives the CTS packet from the receiver. Afterwards, the node switches to the transmit state to send the data packet and switches back to the receive state to wait for ACK packet from the receiver. Finally, after having received the ACK packet, the sensor node goes to sleep by turning the transceiver off at $t = 277$ ms.

3.4 T-MAC

The Timeout-MAC (T-MAC) [18] protocol is a synchronized MAC protocol similar to S-MAC. It uses a listen/sleep scheme which is somewhat adaptive to the current network load. The goal of the protocol is to reduce idle listening time while allowing a higher throughput and lower latency than the S-MAC protocol. A reduced idle listening time is mainly achieved by combining the SYNC and the RTS/CTS phase into one single active phase, which allows for shorter duty cycles. The higher throughput and lower latency is a result of allowing multiple sources to send packets during the same listen/sleep interval, hence allowing a packet delivery over multiple hops to reach its destination in one interval instead of multiple intervals, as it would be the case with S-MAC.

3.4.1 T-MAC Protocol Operation

T-MAC has a fixed wake-up cycle like S-MAC. However, instead of using two separate active phases for synchronization (SYNC) and data transmission (RTS/CTS), it uses just one active phase which handles both synchronization and data transmissions. This is particularly useful in cycles where there is no synchronization performed and allows for shorter duty cycles in contrast to S-MAC. The SYNC and RTS/CTS scheme used by T-MAC are identical to the respective schemes used by S-MAC (cf. Section 3.3.1, 3.3.1).

Timeout Access

A fundamental difference to the S-MAC protocol is that the T-MAC protocol allows for multiple transmissions in a single listen/sleep cycle. In S-MAC, if there is more than one node intending to send data, the nodes must contend for the medium in the RTS/CTS phase. The node winning the contention may send the data, and the losing node must go to sleep until the listen/sleep interval is over and may only contend for the medium in the next RTC/CTS phase. T-MAC on the other hand, allows a node who loses the first contention to contend for the medium in the same active period, right after the other node has finished transmitting. This is realized with a set of rules and a time *TA* which defines the shortest active time in each interval. Figure 3.12 shows the periodic active time *TA* which combines the SYNC and the RTS/CTS phase followed by the sleep phase. Every node wakes up when the interval starts and stays awake for at least time *TA*. If a node senses any communication on its radio it stays awake for another time *TA*. If the time *TA* goes by without any transmission on the network, the nodes will go to sleep until the next scheduled time to wake-up. This simple scheme makes it possible for multiple pairs of nodes to communicate in the same listen/sleep period.

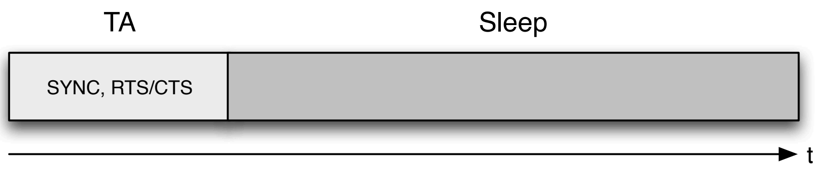

Figure 3.12: T-MAC states

Overhearing Avoidance

A further option in T-MAC is overhearing avoidance just like in S-MAC. This means that nodes may go to sleep when they overhear RTS/CTS packets and wake-up when the current transmission is expected to be over. The NAV, which is contained in the RTS/CTS packet, informs the nodes of how many data packets that the source wishes to transmit. Figure 3.13 shows a scenario with two communicating nodes B and C. It also contains two other nodes A and D which are located in the transmission range of B and C. Overhearing avoidance is deactivated for node A and activated for node D. Consequently, node D goes to sleep after it has overheard the RTS packet of node B and wakes-up as soon as it has predicted the end of the conversation between nodes B and C. Node A stays awake and overhears the conversation.

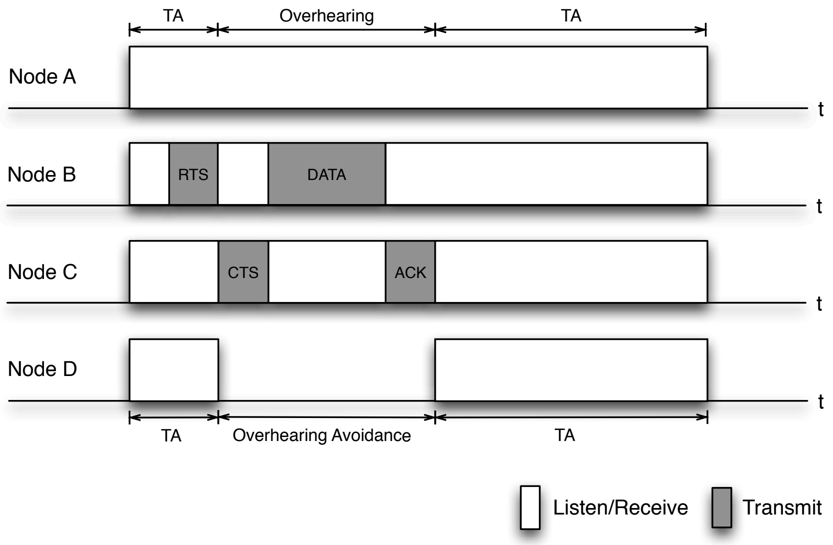

Figure 3.13: Overhearing Avoidance in T-MAC

According to [18], there is a problem that arises when overhearing avoidance is activated. The number of collisions increases as nodes that wake up too late might miss RTS/CTS packets of other nodes and disturb their communication. The authors conclude that it is best to activate overhearing avoidance for little traffic and deactivate it for high traffic where maximum throughput is desired. For the experiments in this thesis, overhearing was permanently deactivated because of stability reasons when sending with higher traffic rates. For lower data rates it was also deactivated, in order to have a common basis for the comparison of the energy usage patterns of the different data rates and protocols.

3.4.2 MSB430 T-MAC Prototype Packet Specification

The packet format used for the T-MAC implementation on the MSB430 sensor nodes is identical to the one used by the S-MAC implementation (see Table 3.4). The only protocol specific field that both implementations utilize is the timestamp/NAV field. The timestamp has the same purpose as in S-MAC. The NAV field, however, has a slightly different purpose in T-MAC, e.g. in the overhearing avoidance option. Nodes overhearing RTS/CTS packets go to sleep for as long as it takes for the sender to transmit as many packets as advertised in the NAV field. Afterwards, the sleeping sensor nodes wake up and stay awake for at least the time *TA*.

3.4.3 MSB430 T-MAC Prototype Parameters

The duration of the *listen/sleep cycle* was set to 1000 ms and the duration of the *SYNC* interval was set to 5000 ms. The values are identical to the ones used for the S-MAC experiments, in order to have similar preconditions for the various protocols. This way the results of the energy measurements of the various protocols can be compared directly and the settings of the parameters play a minor part. The duration of the *TA* phase was set to 100 ms. The calculation of this figure goes hand-in-hand with calculating the duration of the RTS/CTS phase in the S-MAC protocol as explained in Section 3.3.3. The following equation shows the derivation of the figure for the T-MAC protocol.

$$\begin{aligned}\mathrm{TA} &= max(\text{backoff time}) + 2(\text{packet delay}) + \text{adjustment value} \\ &= 39\text{ms} + 36\text{ms} + 25\text{ms} = 100\text{ms}\end{aligned}$$

The active phase of the T-MAC protocol is nothing else than the RTS/CTS phase of the S-MAC protocol with the difference that the phase can also be used to send an receive synchronization packets in addition to RTS/CTS and DATA/ACK packets. Experiments have also shown here that the adjustment value of 25 ms provides stability in a multi-hop environment when nodes transmit at a high data rate.

3.4.4 Typical MSB430 T-MAC Prototype Current Trace

Figure 3.14 is a graphical representation of a sensor node forwarding a packet with T-MAC. The data for this figure was acquired with a SNMD power measurement (see Chapter 4) of a MSB430 sensor node with the transmission power set to the maximum value. The graph shown in the figure is the current draw I of the sensor node at time t.
The sensor node wakes up from the sleep state at $t = 51$ ms, turns on the radio and receives a RTS at $t \approx 90$ ms. It replies with a CTS, as it has not yet received any other RTS/CTS packets, and switches back to the receive state. It receives the packet from the sender at $t \approx 170$ ms and acknowledges the reception with an ACK. Afterwards, the sensor node generates a random

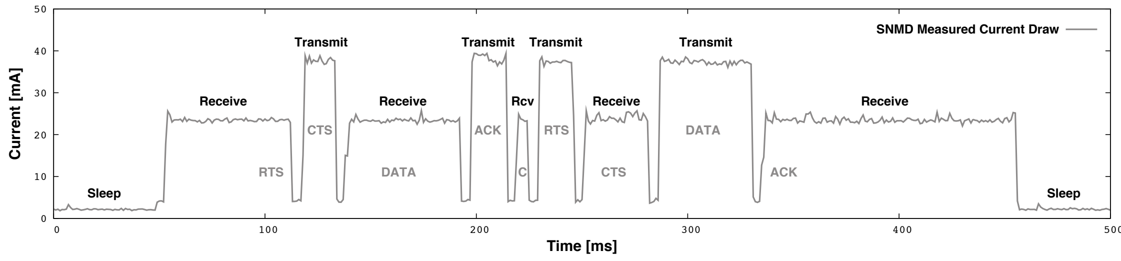

Figure 3.14: Current Draw of a Sensor Node Forwarding a Packet with T-MAC

backoff time to contend for the medium, since it intends to forward the packet. This is depicted in the figure with the letter *C*. The node sends a RTS at $t = 231$ ms, because it has not sensed any traffic during contention. Finally, the node receives a CTS reply, and is able to send the DATA packet at $t = 279$ ms. After it has received the ACK for the DATA packet, it stays awake for time *TA* and goes to sleep at $t = 258$ ms as it has not sensed any traffic during the time *TA*.

3.5 WiseMAC

The WiseMAC protocol [7] is an unsynchronized MAC protocol based on preamble sampling. It aims to lower the energy waste further, by using the preamble sampling technique in addition to learning the wake-up schedules of its direct neighbors. The methodology promises low power consumption with little network traffic, and high energy efficiency in situations of high network traffic. Furthermore, WiseMAC is somewhat adaptive to network load and, like the other MAC protocols presented here, needs no control packets for setting up a network.

3.5.1 WiseMAC Protocol Operation

The preamble sampling technique, which is the basis of WiseMAC, consists of a periodic listen/sleep cycle and lets each node have its own wake-up schedule without any need for synchronization. The only requirement is that all the nodes use the same duration T_W for the wake-up interval. Each node wakes up in each interval and samples the medium for a preamble signal. If the node doesn't sense a signal, it goes back to sleep. The sampling time can be as short as the duration of a modulation symbol (1 byte) which results in a low duty cycle (ratio listen/sleep). If the node senses a signal, it stays awake to receive the packet that follows the preamble. The preamble is a bit pattern sent by a node who wishes to transmit data. In the case of the preamble sampling technique, the sender does not know the next wake-up time of the intended receiver, and the duration of the preamble transmission must be T_W to guarantee that the receiver wakes up sometime during the transmission of the preamble. The intended receiver then acknowledges the received packet with an ACK packet. This process is illustrated in Figure 3.15.
Due to the transmission of long preambles with every packet, the power consumption of the preamble sampling technique is rather high in case of high network traffic. To overcome this problem, WiseMAC introduces the ability to learn the schedules of the direct neighbors and, as

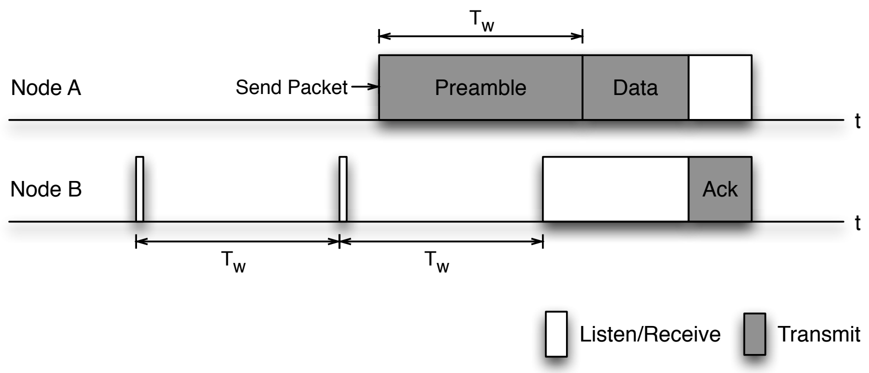

Figure 3.15: WiseMAC scenario before learning sampling schedule of neighbor

a result, shorten the duration of the preamble. The ACK packet, therefore, contains the wake-up schedule of the receiver. With this information, the sender may wait until the receiver awakes and send a short preamble before transmitting the packet. The duration of the preamble T_P and starting time of transmission are calculated by the sender with a tolerance to the clock drift in mind.

$$T_P = min(4\theta L, T_W)$$

L is the time since the last scheduled packet exchange. It is derived by the source with the information that was transmitted in the ACK packet by the receiver. θ is the frequency tolerance of the nodes' quartz based clock. The digitally controlled oscillator (DCO) units of the clocks are driven by a vibrating quartz crystal. The resulting frequency of the resonance of the crystal varies very slightly depending on environmental factors such as temperature, pressure or vibrations. This variance of the frequency results in the drifting of the clocks. If the clock of the receiver is early, the sender has to start transmitting the preamble θL before the scheduled time L. Since the sender's clock may be late and the receiver's clock early, the tolerance has to be increased to $2\theta L$. Finally, if the clock of the sender is early and the clock of the sender is late, the sender would have to start transmitting $2\theta L$ after the scheduled time L. For this reason the duration of the preamble is $4\theta L$ and the transmission will be started at $L - T_P/2$ which designates the center of the expected wake-up time of the receiver. If L is large enough then $4\theta L$ is larger than T_W. In this case the value T_W is used as the duration of the preamble. Figure 3.16 shows the same scenario as in Figure 3.15 but after node A has learned the schedule of node B.

To avoid collisions, WiseMAC incorporates the non-persistent CSMA mechanism. That means that if the sender senses a signal on the medium when attempting to send, it will delay the transmission until the next scheduled wake-up time and try again. Furthermore, in a multi-hop environment the probability of collisions increases as nodes operate as relays for other nodes. WiseMAC tries to minimize the number of collisions by extending the preamble by a randomized duration. This acts as a medium reservation signal, and whoever has the longest preamble wins the contention.

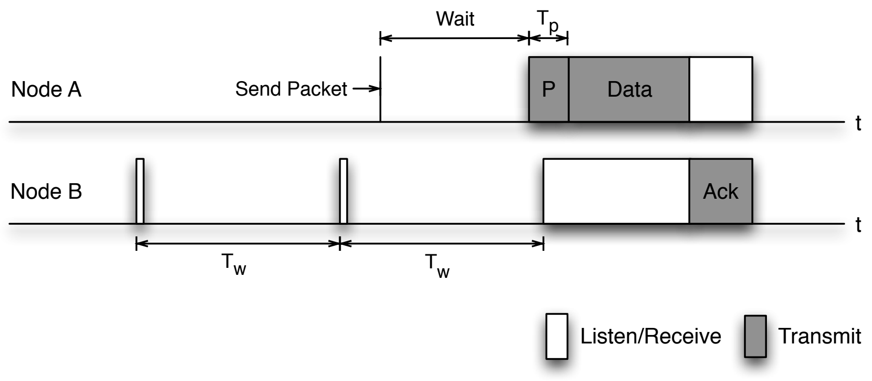

Figure 3.16: WiseMAC scenario after learning the neighbor's sampling schedule of neighbor

In order to decrease the per-packet overhead and increase the throughput of packet bursts, WiseMAC suggests the *more* bit scheme. If a source node has more packets to transmit, it sets the *more* bit. With this bit set, the receiver will stay awake after sending the ACK frame and wait for more data. The sender will start transmitting the next packet immediately after receiving the ACK packet.

3.5.2 MSB430 WiseMAC Prototype Packet Specification

The WiseMAC packet has three fields which are unique in comparison to the other protocols examined in this thesis. The *preamble* field of the WiseMAC has a variable length in contrast to the other protocols. The *type* information and the *more bit* share one field with 6 bits allocated for the *type* and 1 bit allocated for the *more bit*. The *type* information only requires 6 bits because there are less than 64 packet type definitions required for the implementation. The remaining bit of the *type* field may be used by the field *milliseconds*. This field is used by the acknowledgement packets and contains the information of the remaining milliseconds before waking up to sample

Table 3.5: WiseMAC packet format

Field	Bytes	Description
Preamble	**variable**	**Bit sequence for medium reservation**
Start Delimiter	2	Indicates the beginning of the data
Size	1	The size of the packet including payload
Address Target	1	The address of the receiver (0 - 255)
Flags	1	Deactivation of CRC error checking or ACK requests
Address Source	1	The address of the sender (0 - 255)
Number	1	Packets are numbered sequentially
Type/More Bit	**1**	**Packet type and *more* bit**
Milliseconds	**1**	**Milliseconds until next wake-up time**
Payload	variable	The data to be sent
CRC	2	The CRC checksum

the medium. This value may be as large the value T_W. To allow for wake-up cycles higher than 256 ms but no higher than 512 ms, this field had to be extended to 9 bits.

3.5.3 MSB430 WiseMAC Prototype Parameters

The duration of the wake-up cycle T_W was set to 500 ms and the sampling time to 3 ms. It was determined experimentally that a MSB430 requires 3 ms to turn on the radio and reliably sense a preamble byte. A preamble byte alternates between bit values and may therefore have the structure 10101010 or 01010101. Usually, the drift of digital crystal oscillators is no more than $2 \cdot 10^{-5}$ sec/sec [1]. However, the value of $3 \cdot 10^{-5}$ sec/sec was chosen for the frequency tolerance θ because it is assumed a more realistic choice for the hardware clock of the nodes. The notion of a minimum duration preamble was used, as it is possible for $4\theta L$ to have a value of less than 1ms. This is not feasible since the resolutions of the timers of the MSB430 sensor nodes do not allow values smaller than milliseconds. Tests showed that a preamble with a minimum duration of $8 \text{ ms} + \theta$ provides enough headroom for the receivers to wake-up on time in a constant fashion.

3.5.4 Typical MSB430 WiseMAC Prototype Current Trace

Figure 3.17 is a graphical representation of a sensor node forwarding a packet with WiseMAC. The data for this figure was acquired with a SNMD power measurement (see Chapter 4) of a MSB430 sensor node with the transmission power set to the maximum value. The graph shown in the figure is the current draw I of the sensor node at time t.

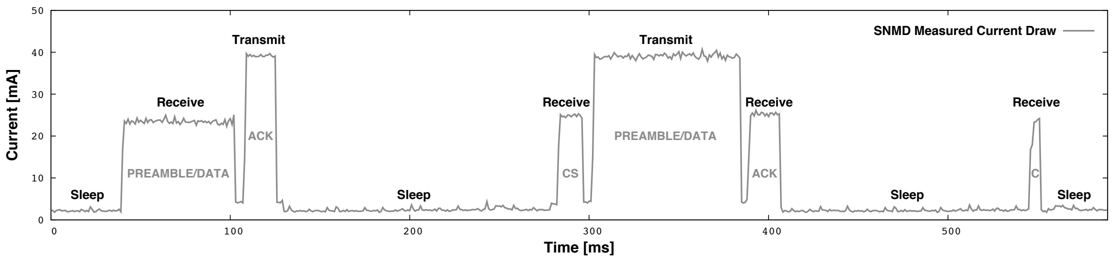

Figure 3.17: Current Draw of a Sensor Node Forwarding a Packet with WiseMAC

The sensor node wakes up from the sleep state at $t = 41$ ms. It senses a preamble and stays awake to receive a DATA packet. After the reception, it sends an ACK packet, derives the wake-up time of the target node (based on its wake-up table) and goes back to sleep. The node wakes up at $t = 284$ ms and listens for potential traffic on the medium (denoted with the letters *CS* in the figure). After it has sensed no traffic, it commences with transmitting the preamble at the derived wake-up time, followed by the packet data. The sensor node then switches to the receive mode and receives the ACK packet, which signifies a successful transmission. Afterwards, the sensor node turns off the radio and only turns it back on at $t = 548$ ms, to sample the medium for a preamble. Since there is no preamble present, it can go back to sleep after 3 ms.

Chapter 4

Experimental Evaluation of Energy Estimation Models

Section 4.1 of this chapter describes the hardware setup used to conduct the experiments. It shows how the physical measurements of the power consumption were made in parallel to the energy estimations and how we derived the total energy consumption for the former. It illustrates further experiment details such as the total number of different sensor nodes used or the total number of tests conducted.
In Section 4.2, we present the first results in the form of an energy consumption comparison between the four tested MAC protocols.
In Section 4.3 we examine the deviations of the power consumptions among different sensor nodes. The section contains the comparison of the power consumptions of 8 different sensor nodes and 4 different MAC protocols used to quantify the deviations. The purpose of this section is to examine the maximum possible energy estimation accuracy of an energy estimation model that relies on the predetermination of the current draw of the various states of a sensor node.
Section 4.4 presents an energy estimation model that uses three states to represent the possible software and hardware parameter configurations in terms of the current draw. It explains the implementation of the prototype on the MSB430 sensor nodes and provides two methods for calibrating the parameters of the model.
Section 4.5 introduces an energy estimation model that incorporates state transitions in addition to the three states of the previous model. The reason for having developed the *three states model with state transitions* was to examine the impact of a higher level of granularity on the accuracy. The section also contains the implementation specifics of the prototype and explains how the calibration methods of the previous model can be adapted to the new model.
In Section 4.6, we compare the estimation accuracy of the models depending on the calibration methodology and the properties of the dataset used for calibration.
Finally, Section 4.7 describes the process of adding sensors to the energy estimation models with the help of the onboard SHT11 sensor and establishes the estimation accuracy of the models including network and sensor activity.

4.1 Experiment Setup

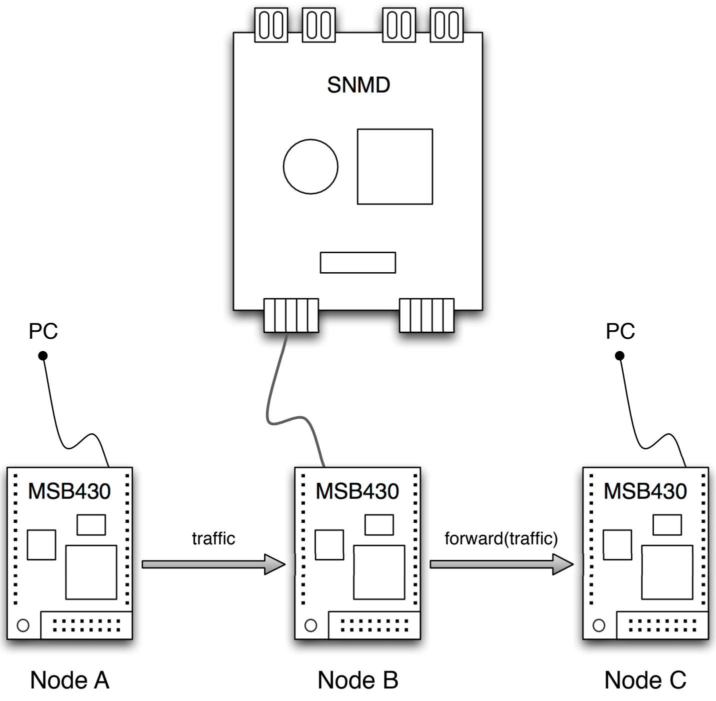

Figure 4.1: Experiment Setup

The general idea for the experiments was to physically measure the power consumption of a sensor node, as it receives and sends traffic at a constant bitrate (CBR) with various MAC protocols, and at the same time have the same node estimate the energy consumption during the whole experiment. Having the physical measurements from the SNMD and the energy estimation from the sensor node would allow the evaluation of the estimation accuracy by quantifying the error rate.

The physical power consumption measurements of the MSB430 sensor nodes were performed using a Sensor Node Management Device (SNMD) (see 2.1.2). To avoid any potential measurement deviations resulting from electrical tolerances of the SNMD components, the same SNMD was used for all experiments in this thesis. The power measurement setup, shown in Figure 4.1, consisted of three MSB430 sensor nodes and a SNMD. Sensor node B was connected to the SNMD for measuring the power consumption, and the remaining two sensor nodes A and C were connected to a desktop PC for controlling the experiment and obtaining energy estimation information from node B. In order to distinguish the transmission state from the reception state, the software configurable transmission power of the CC1020 transceiver has been set to the maximum value for all sensor nodes in the tests.

Node A was programmed to send CBR traffic to node B at a certain data rate. Node B had the task to forward the traffic to node C and add a payload to the packets with the value for its estimated power consumption. Next, node C printed the energy estimation value that it has received from node B to the console via the serial interface. Finally, the PC stored these values in a trace file for later cross-comparison with the physical measurements. Granted, it would have been easier and more accurate to obtain the values for the estimated power consumption directly from node B via its serial interface. However, this was not possible because if node B were connected to a PC through the USB interface, it would have drawn a small amount of current from the USB source and this would have lead to wrong results in the power consumption measurements. The same applied to the batteries, which also had to be removed for the tests. Instead, the negative as well as the positive terminal, located inside the battery compartment, were connected to the appropriate pins of the SNMD, which acted as the power source. The programmable sampling frequency of the SNMD was set to 1000 Hz. This resolution was sufficient as a higher sampling frequency only had a marginal impact on the resulting accuracy. Each sample of the SNMD consisted of the current draw I and the supply voltage V and was sent to the PC via the serial interface where it was stored in a trace file for evaluation.

We have shown graphical representations of trace files excerpts for every MAC protocol in Chapter 3. The excerpts were chosen to highlight the typical properties of the protocols. We were able to discuss the details of the depicted graphs and derive the meaning of each significant change of the current draw. This shows that the described methodology for measuring the current draw can be a powerful tool for debugging and establishing the correctness of the protocol implementations.

With the experiment setup, we have conducted a total of 780 tests to compare the energy consumption of the various MAC protocols, analyze the deviations among sensor nodes and measure the accuracy of the energy estimation methodologies. Each test was 600 seconds long, amounting to a continuous sensor node runtime of 130 hours for all tests. A test consisted of node A sending a constant number of packets per second to node B, which forwarded the packet to node C upon arrival of each packet, see Figure 4.1. The size of each packet was roughly 50 bytes (approximately 10 bytes header and 40 bytes payload), depending on the header size of the MAC protocol used for the experiment. We have used 8 different MSB430 sensor nodes in the place of node B and 9 different data rates ranging from 1 packet every 100 seconds to 2 packets every second. Furthermore, we have conducted each experiment with 4 different MAC protocols. Table 4.1 shows the number of experiments that were conducted with each packet rate and sensor node for evaluating the energy estimation models and MAC protocols. The

Table 4.1: Number of Experiments with Sensor Nodes I - VIII with Respect to the Packet Rate

packet rate [packets/s]	node I [no. tests]	node II [no. tests]	nodes III - VIII [no. tests]
0.01	40	20	0
0.05	40	20	20
0.10	40	20	0
0.20	40	20	0
0.50	40	20	20
0.75	40	20	0
1.00	40	20	0
1.50	40	20	0
2.00	40	20	0

first column contains the packet rate information (0.01-2.00 packets/s) and the remaining three columns contain the number of experiments that were conducted with the respective sensor nodes (I-VIII). E.g. there were a total of 40 tests (10 tests per MAC protocol) conducted using node I with a data rate of 0.2 (1 packet every 5 seconds).

4.2 Energy Consumption Comparison of MAC Prototypes

With the acquired trace files from the SNMD power measurements, it is simple to derive the power consumption of the sensor nodes. The electrical power P at time t is derived as shown in Equation 4.1. It is measured in watts and equals the voltage V multiplied by the current I. Both of these values, I and V, are provided in the trace files in a resolution of 1000 Hz.

$$P(t) = V(t) * I(t) \tag{4.1}$$

The energy consumption E is measured in Joules and equals one Watt second (*Ws*). It can be obtained from the equation shown in 4.2, where P is the electrical power at time t. Informally, the energy consumption E corresponds to the area below the current draw, as pictured in one of the graphical representations of trace files excerpts (Figures 3.3, 3.11, 3.14 and 3.17), multiplied with the supply voltage V.

$$E_{[t_{\text{start}},t_{\text{end}}]} = \int_{t_{\text{start}}}^{t_{\text{end}}} P(t)dt \tag{4.2}$$

Figure 4.2 shows the relationship of the data rate and power consumption of the tested MAC protocols. The test data for each of the energy consumption graphs (CSMA, S-MAC, T-MAC

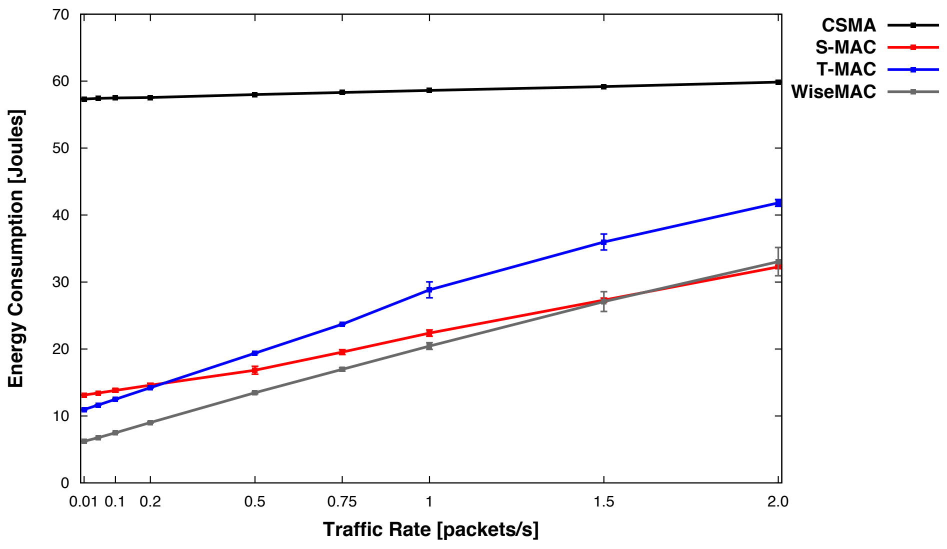

Figure 4.2: MAC Protocols Power Consumption Comparison

and WiseMAC) consisted of 10 tests for each data rate measured with one sensor node. As expected, the power consumption of CSMA was the highest of all and increased only very little with an increasing data rate. The increase was a mere 4% from the lowest to the highest data rate. As a matter of fact, the only reason for this increase was that we have set the transmission power of the transceiver to the maximum value. This had the consequence that the current draw when transmitting was higher than when receiving or idle listening. It is possible to observe the reverse effect, when setting the transmission power to the lowest setting. In that case, the current draw of transmitting is lower than that of receiving which leads to a decrease in the energy consumption with an increasing data rate.
The behavior of the three E^2-MAC protocols differs in a way that the power consumption is not only dependent on the data rate and the ratio current draw(transmit)/current draw(receive/listen), but also on how much time the sensor nodes spend idle listening, overhearing and sleeping. This can be seen clearly when comparing the power consumption of the MAC protocols at the lowest data rate. At the data rate of 1 packets every 100 seconds, the highest energy consumption is exhibited by CSMA because it never turns off the transceiver. The lowest energy consumption at this rate is exhibited by WiseMAC as it has the lowest duty cycle at 0.6% with the transceiver only being turned on for 3 ms every 500 ms when the node is not sending nor receiving data. T-MAC comes in at a second place with a duty cycle of 10% and S-MAC at third with a duty cycle of 15%. As the data rate increases, so does the influence of other protocol specific behavior on the power consumption. At the highest data rate, the power consumption of the S-MAC protocol is on par with that of WiseMAC and that of T-MAC is nearly 30% higher. The reason behind this is that unlike T-MAC, S-MAC and WiseMAC prevent the costly overhearing of traffic for other sensor nodes. For S-MAC this comes with the cost of a low end-to-end delay, as the packets are able to travel a maximum of 1 hop per wake/sleep cycle. This limitation does not apply to T-MAC, as the protocol allows multiple transmission in a single wake/sleep cycle. The number of hops the packets are able to travel in a single wake/sleep cycle with T-MAC is limited solely by the size of the packets and the duration of the wake/sleep cycle. Apart from CSMA, T-MAC exhibits the shortest end-to-end delay at the highest data rate as bursts of 2 packets can easily travel 2 hops in one wake/sleep cycle. The end-to-end delay of the WiseMAC protocol is largely dependent on the distribution of the wake-up times of the individual nodes along the path. The further apart the wake-up times of senders and receivers, the higher is the per-hop delay. Since the wake-up times of the nodes are set randomly at the start up of each node, the end-to-end delay of WiseMAC varies in every run and may be as high as with the S-MAC protocol or as low as with the T-MAC protocol.

4.3 Deviations Among Different Sensor Nodes

Previous studies [14] [10] have observed variations of the power consumption among different sensor nodes of the same type. The total energy consumption of a sensor node depends on the amount of current drawn by each of the various electrical components. Since these components may have a varying current draw, which may be a result of manufacturing tolerances [10], the total power consumption may vary from node to node. Naturally, the tolerances of these components fabricated in highly precise manufacturing facilities are very low. The sum of all

components though, may lead to a total deviation of up to a few percent among different nodes. It is also known that the power consumption of electronic components is affected by the temperature. The temperature can be ruled out as a cause for the deviations in our experiments, since all our experiments were performed under the same environmental circumstances (lab conditions at room temperature).
Any mechanism that is purely software driven to estimate the energy consumption of a device will depend on the prior knowledge of the current draw of individual components or groups of components. The values for the current draw can be determined through the statistical analysis of current draw measurements of a single sensor node, or through the analysis of measurements of multiple nodes. In addition to the imperfections of a software-based energy estimation model, both methodologies to determine the values for the current draw of the components inherently lead to inaccuracies in estimating the power consumption. Obtaining the values with just one device leads to a more accurate energy estimation for that particular device, but deteriorates the results of other devices' energy estimations. When obtaining the values with multiple devices, the accuracy for the single device will degrade, but the average accuracy for all other devices will improve.

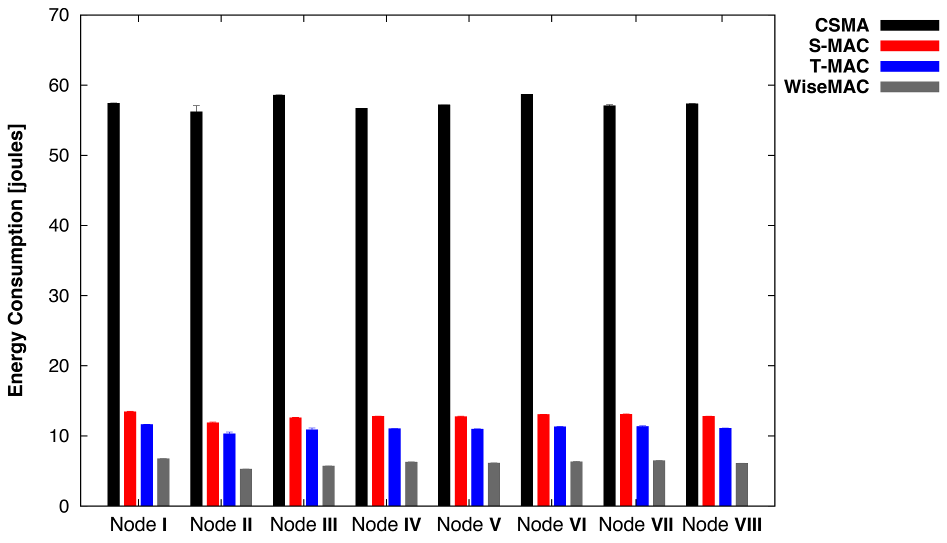

Figure 4.3: Power Consumption Deviation Among Different MSB430 Sensor Nodes

Figure 4.3 illustrates to what extent the physically measured power consumption of 8 different MSB430 sensor nodes varies. Each graph in the Figure represents the average power consumption of a sensor node and MAC protocol of 10 test runs with data rates of 0.05 and 0.5 packets/s. The details of the experiment setup and test runs are explained in Section 4.1. The standard deviations between the individual runs are also included in the Figure, they are very small however and barely visible. This is an indication that the small variance between the runs of the same experiment may not constitute the rather large difference of the power consumption between the different nodes. Indeed, the standard deviation between all sensor nodes for the tests with the T-MAC protocol is 0.4 joules (3.6%), and the standard deviation of the individual nodes averages

at only 0.09 joules (0.8%) per node. The same observation can be made for CSMA with 0.9 joules (1.5%) versus 0.1 joules (0.2%), S-MAC with 0.5 joules (3.5%) versus 0.06 joules (0.5%) and WiseMAC with 0.5 joules (7.5%) versus 0.06 joules (0.9%). To strengthen the assumption that this observation is not a result of coincidence we have compared the power consumption of all node pairs, and tested for statistical significance of the differences with a two-sided t-test. Table 4.2 shows the percentages of the power consumption deviations followed by the confidence level (in brackets) of all node pairs and MAC protocols tested. The confidence levels in the table denote the likelihood that the difference in power consumption of two sensor nodes is not due to random chance. For instance, there is a 99.98% chance that the 1.8% deviation of the S-MAC power consumption of node III and node IV is significant and not due to coincidence. A total of 90% of all node pairs and MAC protocol combinations exhibit a confidence level higher than 95%. The majority of the node pairs that produce a confidence level lower than 95% feature a similar energy consumption pattern and require more test data to conclude a significant difference in terms of the energy consumption. As can be seen on the table, the E^2-MAC protocols produce higher deviations than the CSMA protocol. We attribute this observation to the higher variation of the test runs of each node and to the fast switching of the radio operation modes. It is possible that the latter temporarily affects the behavior of the active circuit elements on the board, and has an impact on the energy consumption. Both of these attributes peak in the WiseMAC protocol, which exhibits the fastest radio operation mode switches (preamble sampling requires only a very quick listening periods) and the highest variation of energy consumption of an individual sensor node. Evidently, the WiseMAC protocol also produces the largest power consumption deviations between the different sensor nodes.
Because of the low variation between the test runs, the CSMA protocol is the most suited when attempting to quantify the deviation of the power consumption among different sensor nodes independently of the MAC protocols. It produces the most conservative results with the lowest deviation being 0.1% between nodes I and VIII and the highest being 4.4% between nodes II and VI. The null hypothesis that the means of these power consumption values are equal and the difference is coincidental can be rejected with a confidence level of 96.5% for the former and 99.98% for the latter.

4.4 Three States Model

In a general computing context, a state refers to a unique configuration of the software and hardware parameters. For software-based energy estimation models, the term state may also specify multiple software and hardware parameter configurations, which exhibit identical or similar current draws. The basic scheme of software-based energy estimation for sensor nodes presented here is to identify a set of these states which are able to represent all possible software and hardware parameter configurations in terms of the current draw. On the sensor node, a software extension will be responsible of keeping track of the times in which these states are active, henceforth called the retentions of the states. In the end, the total power consumption may be derived from the retention of each state together with the predetermined current draw of each state.
The Three State Model as described in [10], identifies 3 different states for a sensor node. We

Table 4.2: Power Consumption Deviation (%) and the Respective Confidence Levels (%)

CSMA	node II	node III	node IV	node V	node VI	node VII	node VIII
node I	2.1 (98.71)	2.0 (100)	1.2 (100)	0.4 (100)	2.2 (100)	0.6 (99.96)	0.1 (96.5)
node II		4.2 (99.98)	0.9 (77.65)	1.7 (96.75)	4.4 (99.98)	1.5 (94.77)	2.0 (98.28)
node III			3.3 (100)	2.4 (100)	0.2 (100)	2.6 (100)	2.2 (100)
node IV				0.8 (100)	3.5 (100)	0.7 (99.98)	1.1 (100)
node V					2.6 (100)	0.2 (90.5)	0.3 (99.99)
node VI						2.8 (100)	2.4 (100)
node VII							0.5 (99.81)

S-MAC	node II	node III	node IV	node V	node VI	node VII	node VIII
node I	12.9 (100)	6.7 (100)	4.8 (100)	5.3 (100)	2.8 (100)	2.7 (99.99)	5.0 (100)
node II		5.8 (100)	7.8 (100)	7.3 (100)	9.9 (100)	10.0 (100)	7.6 (100)
node III			1.8 (99.98)	1.4 (99.29)	3.8 (100)	3.9 (100)	1.6 (99.94)
node IV				0.4 (81.89)	2.0 (100)	2.1 (100)	0.2 (72.51)
node V					2.4 (100)	2.5 (99.99)	0.2 (54.93)
node VI						0.1 (37.94)	2.2 (100)
node VII							2.3 (100)

T-MAC	node II	node III	node IV	node V	node VI	node VII	node VIII
node I	13.1 (100)	6.9 (99.98)	5.4 (100)	5.9 (100)	2.9 (100)	2.4 (99.98)	4.8 (100)
node II		5.8 (99.26)	7.3 (99.97)	6.8 (99.95)	9.9 (100)	10.4 (100)	7.9 (99.98)
node III			1.4 (79.16)	0.9 (57.75)	3.9 (99.34)	4.4 (99.54)	2.1 (91.33)
node IV				0.5 (98.88)	2.4 (100)	2.9 (99.99)	0.6 (100)
node V					3.0 (100)	3.4 (100)	1.1 (99.99)
node VI						0.4 (67.43)	1.8 (100)
node VII							2.2 (99.97)

WiseMAC	node II	node III	node IV	node V	node VI	node VII	node VIII
node I	28.0 (100)	18.4 (100)	7.7 (100)	10.0 (100)	7.2 (100)	4.4 (100)	11.1 (100)
node II		8.3 (100)	18.9 (100)	16.5 (100)	19.5 (100)	22.8 (100)	15.4 (100)
node III			9.8 (100)	7.6 (100)	10.4 (100)	13.4 (100)	6.5 (100)
node IV				2.1 (99.74)	0.5 (50.89)	3.2 (99.99)	3.1 (99.99)
node V					2.6 (99.67)	5.4 (100)	1.0 (96.86)
node VI						2.7 (99.81)	3.6 (99.97)
node VII							6.4 (100)

refer to them as the *sleep* state, the *receive* state and the *transmit* state. The *sleep* state merely implies that the transceiver component is turned off but does not specify the activity of the CPU. The *transmit* state is active when the sensor node is transmitting data and the *receive* state is active when the sensor node is idle listening or receiving data. The reason for this combination is that there is no clear distinction between a MSB430 sensor node that is listening to the channel and a node that is actually receiving data in terms of current draw. However, we have seen cases of MSB430 sensor nodes that exhibit a slight variation of the current draw when idle listening and receiving data. Since this is only present in rare cases of the MSB430 sensor nodes, a distinction of this would only result in accuracy gains for those rare cases of sensor nodes and only if they are calibrated individually.
Figure 4.4 illustrates how the current draw of the trace file excerpt shown in Figure 3.11 is approximated by the Three States Model. The current draw values of the individual states in

the figure were predetermined by averaging the respective values of the physical measurements. The visual representation of the current draw demonstrates the information loss in contrast to the available information from the physical measurement.

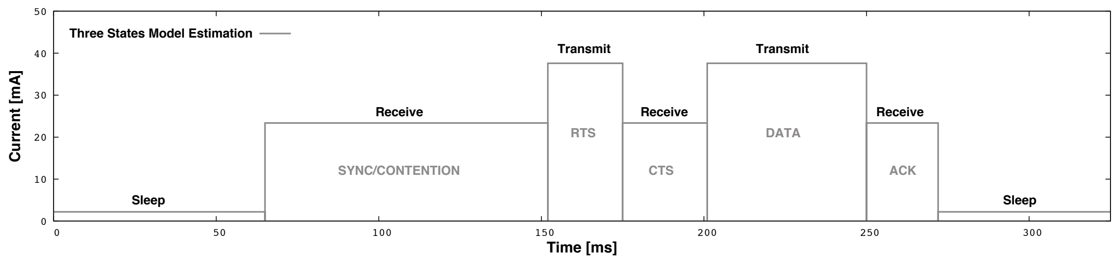

Figure 4.4: Current Draw as Modeled with the Three States Model

4.4.1 Three States Model Prototype Implementation

The implementation of the Three State Model on the ScatterWeb operating system consists of a library `libEnergyEstimation` which is integrated into the operating system in the same way as the radio device driver. The library contains the variables t_{sleep}, t_{receive} and t_{transmit} which represent the retentions of the respective states, e.g. a value of 25 ms for t_{transmit} specifies that the transceiver of the sensor node has been transmitting for a total of 25 ms since the initialization of the energy estimation mechanism. The library also contains a few other variables which are necessary for the update mechanism of these retention values. The variable S retains the current state of the machine (sleep, receive, or transmit) and the variables T_{start} and T_{end} store the values for the start and end time of the current state S. The values of those variables are initialized with a call to the function.

```
EnergyEstimation_init();
```

This function resets all the state retention values to 0 ms, sets the value for S to the current state of the sensor node and T_{start} to the current time. Having an initialization function which can be called at any time is convenient for restarting the energy estimation mechanism in order to estimate the energy of a sensor node for certain time periods only. During this estimation period, the values of the retention times t_{sleep}, t_{receive} and t_{transmit} have to be updated in accordance with the actual behavior of the sensor node. To update the values, the library provides an interface with a few function calls that have to be added to the right places in the sensor node operating system. The process involves the detection of transceiver state switches which are most likely found in the network protocol implementation of the sensor node. In the case of ScatterWeb they can be found in the C file `ScatterWeb.Net.c`. One of the following three function calls has to be added to the switching locations of the code.

```
EnergyEstimation_switch_to_NET_SLEEP();
EnergyEstimation_switch_to_NET_RECEIVE();
EnergyEstimation_switch_to_NET_TRANSMIT();
```

Whenever one of these functions are called, the variable T_{end} is set to the current time and the retention of the current state S is updated with $t_S = t_S + T_{\text{end}} - T_{\text{start}}$. Afterwards, the value of S is set to the new active state of the sensor node and T_{start} to the current time. Proper integration of the function calls into the network protocol implementation of the sensor node results in continuous updates to the retention times of the states as the sensor node physically switches between sleep, receive and transmit. With the up-to-date retention times, it is possible to obtain the value for the power consumption at any time. The following function call returns a floating point value which represents the value of the energy consumption in joules since the initialization of the estimation model.

```
EnergyEstimation_getConsumption();
```

The execution of this function has similar consequences as a state change. Just like in a normal state change, T_{end} is set to the current time and the retention of the current state is updated with $t_S = t_S + T_{\text{end}} - T_{\text{start}}$. Afterwards, T_{start} is set to the current time. In contrast to a state switch, however, the value for the current state S does not change. This ensures that the latest active state is included in the calculation of the power consumption. The derivation of the energy consumption E is shown in Equation 4.3 where t_{sleep}, t_{receive} and t_{transmit} are the retention times as mentioned above and P_{sleep}, P_{receive} and P_{transmit} are the predetermined power rates of the respective states measured in watts.

$$E = P_{\text{sleep}} t_{\text{sleep}} + P_{\text{receive}} t_{\text{receive}} + P_{\text{transmit}} t_{\text{transmit}} \tag{4.3}$$

The electrical power of a state P_{state} is the result of the supply voltage V multiplied with the current draw I of the respective state as shown in Equation 4.4. The derivation is similar to the one shown in Equation 4.1 for the calculation of the electrical power from physical measurements. The main difference is that unlike $P(t)$, which has variable values and represents the measured electrical power at a particular time t, P_{state} has a constant value which represents the power of a state in general. Consequently, the parameter I_{state} is constant for each state of the Three States model.

$$P_{state} = V * I_{state} \tag{4.4}$$

4.4.2 Parameter Determination of the Three States Model with Average Values from Power Measurements

In a first attempt, the calibration of the three state model was done by determining the different electrical power values P_{sleep}, P_{receive} and P_{transmit}. One possibility to obtain the average values is to analyze the trace files of a number of physical power consumption measurements. A trace file, as explained in Section 4.1, contains a value for the current I and the voltage V for every millisecond. The idea is to identify the current values that belong to a certain state and obtain a figure I_{state} that represents the current draw of a state by averaging these values. Figure 3.11 shows that with the transmit power set to the highest setting, the current draws of the different states can be easily distinguished visually. We have determined a range of 0 - 3 mA for I_{sleep}, 20 - 30 mA for I_{receive} and 31 - 45 mA for I_{transmit}. The following algorithm in pseudo code illustrates how the average values for the states are calculated. The variable i

represents the current draw I at time t. The three if-statements assign the current draw to the correct state (*receive*, *transmit*, *sleep*) by adding the value of i to temporary variables of the respective states (I_r, I_t, I_s), while counting the occurrences of these additions (r, t, s). Finally, the average current draw of each state (I_receive, I_transmit, I_sleep) is calculated by dividing the temporary current draw values by their respective occurrences.

```
for all current values $i$ do
    if $i \leq 3$ then
        $I_\mathrm{s} \leftarrow I_\mathrm{s} + i$
        $s \leftarrow s + 1$
    else if $20 \leq i \leq 30$ then
        $I_\mathrm{r} \leftarrow I_\mathrm{r} + i$
        $r \leftarrow r + 1$
    else if $31 \leq i \leq 45$ then
        $I_\mathrm{t} \leftarrow I_\mathrm{t} + i$
        $t \leftarrow t + 1$
    end if
end for

$I_\mathrm{sleep} \leftarrow I_\mathrm{s}/s$
$I_\mathrm{receive} \leftarrow I_\mathrm{r}/r$
$I_\mathrm{transmit} \leftarrow I_\mathrm{t}/t$
```

The voltages in the trace files are nearly constant and not dependent on the sensor node state. Therefore, we calculated the average supply voltage across all input values to obtain one voltage figure for all states. Having aquired the value for the current draw I_state for each state and the voltage V, the electrical power of a state P_state may be calculated by multiplying the current draw of the respective state with the voltage as shown in Equation 4.4.

4.4.3 Parameter Determination of the Three States Model with Regression Analysis of Experiment Results

Another possibility to determine the parameters of the three state model is by analyzing the relationship of the variables t_sleep, t_receive and t_transmit and the energy consumption E. The values of these variables are obtained through concurrent physical measurements and estimations of the energy consumption of sensor nodes as explained in Section 4.1. The relationship of the variables may be analyzed by means of a linear regression model and the unknown parameters P_sleep, P_receive and P_transmit estimated with ordinary least squares (OLS) regression analysis.
The dataset consists of n experiments $\{E_i, t_{\mathrm{sleep}_i}, t_{\mathrm{receive}_i}, t_{\mathrm{transmit}_i}\}_{i=1}^n$, where E_i is the observed response (energy consumption obtained through hardware-based measurement) and the three remaining variables t_sleep, t_{receive_i} and t_{transmit_i} are the observed regressors. The observed responses and regressors represent a system of linear equations (an overdetermined system if there are more equations than unknown parameters) with n linear equations and 3 unknown

parameters. The system is shown in Equation 4.5,

$$X\beta = y \tag{4.5}$$

where β is a vector which contains the unknown parameters β_1, β_2 and β_3, X is a n-by-3 matrix that contains the observed regressors and y is vector that contains the observed responses.

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} t_{\text{sleep}_1} & t_{\text{receive}_1} & t_{\text{transmit}_1} \\ t_{\text{sleep}_2} & t_{\text{receive}_2} & t_{\text{transmit}_2} \\ \vdots & \vdots & \vdots \\ t_{\text{sleep}_n} & t_{\text{receive}_n} & t_{\text{transmit}_n} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

Since the system usually has no solution, the goal of the linear regression analysis is to find values for the coefficients β, which provide the *best* fit for the equations. The OLS approximation model, shown in Equation 4.6, determines the coefficients with the *best* fit by solving the quadratic minimization problem. A prerequisite of the minimization problem is that the n columns of matrix X are linearly independent.

$$\beta = (X^T X)^{-1} X^T y \tag{4.6}$$

The result of this calculation is a set of values for the unknown parameters β_1, β_2 and β_3 which minimize the sum of the squares of errors made in solving the Equation 4.7 with all the values from every experiment in the dataset. The values β_1, β_2 and β_3 and Equation 4.7 correspond to the values P_{sleep}, P_{receive} and P_{transmit} and Equation 4.3 for deriving the power consumption.

$$E = \beta_1 t_{\text{sleep}} + \beta_2 t_{\text{receive}} + \beta_3 t_{\text{transmit}} \tag{4.7}$$

4.5 Three States Model with State Transitions

The high resolution of the graphical representation of the power consumption shown in Figure 3.11 makes it apparent that when a sensor node switches from one state to another, the current draw does not instantly reflect the value of the new state. Instead, the current draw increases or decreases for a short period of time before leveling out to the current draw of the new state.
For this reason we have developed the so-called *three states model with state transitions* which allows us to examine the impact of the state transitions on the energy estimation accuracy. The

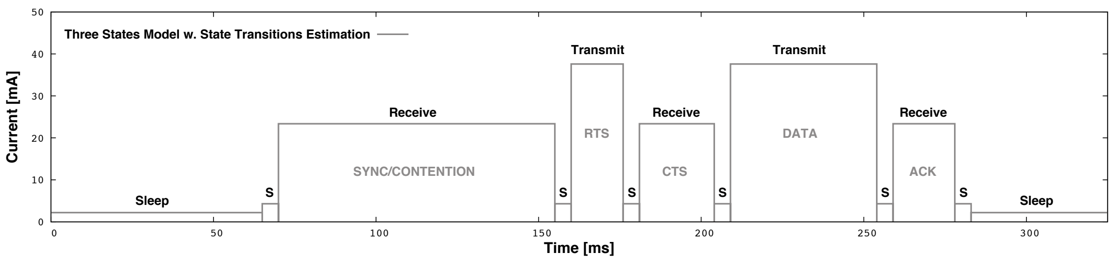

Figure 4.5: Current Draw as Modeled with the Three States Model with State Transitions

new model incorporates a new state that models the current draw of the state transitions. We refer to this state as the *switch* state as it is active with every occurrence of a state transition. The *switch* state is complementary to the three existing states *receive*, *transmit* and *sleep* of the *three states model* and results in a model with a higher level of granularity than the *three states model*. Figure 4.5 shows how the current draw of the physical measurement in Figure 3.11 is approximated by the *three states model with state transitions*.

4.5.1 Three States Model with State Transitions Prototype Implementation

The implementation of the *three states model with state transitions* on the ScatterWeb operating system is an extension of the *three state model* described in Section 4.4.1. The integration of the model into the operating system is identical to the *three states model*. A library, which features the same interface with the same functions as the *three states model*, is added to the operating system. Also, the same function calls to log the retention times t_{sleep}, t_{receive} and t_{transmit} of the three states are added to the code of the network protocol implementation of the sensor node. From the viewpoint of the network protocol the *three states model with state transitions* is no different than the ordinary *three states model*. The differences of the two models are buried inside the functions which log the retention times and calculate the energy consumption. To accommodate the state transitions, the library contains additional variables for logging and new constants that need to be determined prior to estimating the energy.
The additional variables x_{toSleep}, $x_{\text{toReceive}}$ and $x_{\text{toTransmit}}$ disclose the number of times the sensor node has switched to the respective state during estimation. The same functions of the *three states model* that keep track of the state retention times are also responsible of incrementing the respective counters x_{toState}. For instance, $x_{\text{toTransmit}} = 12$ denotes that the sensor node has switched to the transmit state 12 times since initialization of the *three states model with state transitions*.
When the call to calculate the energy consumption is made the state transitions need to be deducted from the respective state retentions as the state transitions are included in the retention times. This requires the prior knowledge of the duration of each state transition. The constants T_{toSleep}, $T_{\text{toReceive}}$ and $T_{\text{toTransmit}}$ hold the predetermined values for the duration of each respective state transition. The formulas for the deductions are depicted in the Equations 4.8, 4.9 and 4.10.

$$t_{\text{sleep}} := t_{\text{sleep}} - x_{\text{toSleep}}T_{\text{toSleep}} \tag{4.8}$$

$$t_{\text{receive}} := t_{\text{receive}} - x_{\text{toReceive}}T_{\text{toReceive}} \tag{4.9}$$

$$t_{\text{transmit}} := t_{\text{transmit}} - x_{\text{toTransmit}}T_{\text{toTransmit}} \tag{4.10}$$

The purpose of the variable t_{switch} is equivalent to the other state retention variables and stands for the total time the sensor node has spent switching from state to state. It can be easily calculated by adding up all the state transition counters multiplied with the respective state transition durations as shown in the following Equation 4.11.

$$t_{\text{switch}} := x_{\text{toSleep}}T_{\text{toSleep}} + x_{\text{toReceive}}T_{\text{toReceive}} + x_{\text{toTransmit}}T_{\text{toTransmit}} \tag{4.11}$$

Finally, the energy consumption may be derived with the help of the electrical power P and the retention time t of the *switch* state in addition to the other retention times and electrical power values already present in the *three states model*. The formula for the calculation is shown in Equation 4.12. The electrical power of the switch state P_{switch} represents the current draw of the sensor node when switching states and must be predetermined in the same way as the other electrical power values.

$$E = P_{\text{sleep}} t_{\text{sleep}} + P_{\text{receive}} t_{\text{receive}} + P_{\text{transmit}} t_{\text{transmit}} + P_{\text{switch}} t_{\text{switch}} \tag{4.12}$$

4.5.2 Parameter Determination of the Three States Model with State Transitions with Average Values from Power Measurements

The calibration of the *three States model with state transitions* involves the determination of the power values P_{sleep}, P_{receive}, P_{transmit} and P_{switch} and the state transition durations T_{toSleep}, $T_{\text{toReceive}}$ and $T_{\text{toTransmit}}$. This can be achieved with a number of trace files obtained through the physical measurements of sensor nodes and some enhancements to the algorithm shown in Section 4.4.2 for finding average values from power measurements. In addition to the ranges found with the calibration of the *three states model*, we have determined the range of 4 - 10 mA for the current draw I of the *switch* state. The determination of the state transition durations is a matter of counting the number of consecutive values which are in the range of the current draw of the *switch* state and lie in between two states. Each value in a trace file represents the current draw of one millisecond since we have set the sampling frequency of the SNMD to 1000 Hz. The total number of these values are kept for each respective state transition type and in the end divided by the number of occurrences of the individual state transitions to gain the average duration. The following algorithm in pseudo code illustrates how the average values for the state transition durations are determined from a number of trace files.
The variable i represents the current draw I at time t. The first if-statement decides if the current draw value is within the range of the *switch* state (4 - 10). The next while-statement counts the number of consecutive current draw values which lie in this range and belong to the same *switch* state. The variable *millis* now corresponds to the duration of the *switch* state. The next three if-statements determine which state (*receive*, *transmit*, *sleep*) is followed by the *switch* state and allow the correct assignment of the duration (*millis*) to the respective *switch* state duration (T_r, T_s, T_t), while counting the number of occurring switches (r, s, t). In the end, the average duration of each *switch* state is calculated by dividing the *switch* state durations with the respective number of occurring switches.
The algorithm is a simplification of the actual algorithm that should be used as there may be outliers in the current draw measurements. Outliers stem from short current draw outbursts in the *sleep* state that fall into the range of the state transitions. Simple pre conditions in the algorithm are able to prevent these outliers from distorting the durations of the state transitions. Through visual examination of various trace files we have determined that a safe range for state transition durations is between 2 and 10 ms.

```
**for all** current values $i$ **do**
  **if** $4 \leq i \leq 10$ **then**
    $millis \leftarrow 1$
    **while** $4 \leq (i++) \leq 10$ **do**
      $millis \leftarrow millis + 1$
    **end while**
    **if** $i \leq 3$ **then**
      $T_s \leftarrow T_s + millis$
      $s \leftarrow s + 1$
    **else if** $20 \leq i \leq 30$ **then**
      $T_r \leftarrow T_r + millis$
      $r \leftarrow r + 1$
    **else if** $31 \leq i \leq 45$ **then**
      $T_t \leftarrow T_t + millis$
      $t \leftarrow t + 1$
    **end if**
  **end if**
**end for**

$T_{\text{toSleep}} \leftarrow T_{\text{s}}/s$
$T_{\text{toReceive}} \leftarrow T_{\text{r}}/r$
$T_{\text{toTransmit}} \leftarrow T_{\text{t}}/t$
```

4.5.3 Parameter Determination of the Three States Model with State Transitions through Regression Analysis of Experiment Results

Linear regression analysis can not be applied to the *three states model with state transitions* in exactly the same way as the regression analysis for the *three states model* as explained in Section 4.4.3. The dependency of t_{switch} on the state transition durations T_{toSleep}, $T_{\text{toReceive}}$ and $T_{\text{toTransmit}}$, which have to be estimated in advance, would lead to a non-linear case of regression analysis. For this reason we have changed the formula for calculating the energy consumption in a way that allows for linear regression analysis and the estimation of the unknown parameters with the OLS method. The adapted formula is shown in Equation 4.13. The reason for replacing the electrical power terms P_{sleep}, P_{receive} and P_{transmit} with β_1, β_2 and β_3 is to emphasize that these do not reflect real world electrical power values in contrast to the original formula.

$$E = \beta_1 t_{\text{sleep}} + \beta_2 t_{\text{receive}} + \beta_3 t_{\text{transmit}} + \beta_4 x_{\text{toSleep}_i} + \beta_5 x_{\text{toReceive}_i} + \beta_6 x_{\text{toTransmit}_i} \quad (4.13)$$

The dataset for the OLS consists of n experiments with the the observed response and the regressors $\{E_i, t_{\text{sleep}_i}, t_{\text{receive}_i}, t_{\text{transmit}_i}, x_{\text{toSleep}}, x_{\text{toReceive}}, x_{\text{toTransmit}}\}_{i=1}^{n}$. The result of applying the OLS method is a set of values for the unknown parameters β_1, β_2, β_3, β_4, β_5 and β_6 that minimizes the sum of the squared errors made in solving the Equation 4.13 for all entries of the dataset.

4.6 Energy Estimation Accuracy Comparison of the Different Energy Estimation Models and Calibration Methodologies

The parameters of the energy estimation models are determined with a number of experiments from one or more wireless sensor nodes and MAC protocols. The result of these experiments is a dataset, which serves as a basis for estimating the parameters through OLS regression analysis or through averaging the power values in the trace files. We refer to a combination of the properties for calibration as the *calibration methodology*. E.g. Two wireless sensor nodes with S-MAC and parameter estimation through OLS regression analysis. An individual experiment is defined by the properties MAC protocol and traffic rate (see Section 4.1).
We have used two disjoint datasets for every evaluation of an estimation model in conjunction with a certain calibration methodology. The dataset used for calibration is referred to as the *training set*, and the dataset used for deriving the accuracy is called the *test set*. As a statistical measure for the accuracy, we calculated the mean absolute error (MAE) [5] of the estimations in relation to the traffic rate. The MAE specifies the average difference of the estimated values to the measured power consumption values of the SNMD. For the experiment results of the models calibrated with OLS regression analysis, we provide the R^2 coefficient as an indication of how well the model fits the dataset. The value of R^2 is always between 0 and 1 with higher values indicating a better fit and the value 1 specifying a perfect fit. Finally, we calculated the mean error across all traffic rates for each estimation model and calibration methodology.

4.6.1 Per-Node Calibration with Multiple MAC Protocols

Per-Node calibration implies that the parameters for the energy estimation model is calibrated on the same sensor node as used for estimating the energy consumption. The term multiple MAC protocols implies that the calibration on the sensor node is performed with a dataset that includes experiments with several different MAC protocols. The practical implication of this calibration methodology is that each node has to be calibrated individually by conducting a number of power measurement experiments with various MAC protocols. Figure 4.6 shows the percentage of the MAE in relation to the traffic rate for the test set which we have used to evaluate per-node calibration with multiple MAC protocols. The training set consisted of a total of 180 experiments. 5 experiments were conducted with the same sensor node (sensor node I, see Table 4.1) for each of the 4 MAC protocols (CSMA, S-MAC, T-MAC and WiseMAC) and each of the 9 traffic rates. The test set also consisted of 180 experiments with the same attributes (5 experiments for each of the 4 MAC protocols) using the same node as used for the experiments in the training set.
The mean error calculated across all traffic rates of the *three states model* calibrated with average values from power measurements is 4.01% with a standard deviation of 3.3%. The mean error of the *three states model with state transitions* using the same calibration methodology is 3.9% with a standard deviation of 4.5%. The model with state transitions exhibits larger errors in experiments with traffic rates below 0.5 packets/s. However, the degree of the errors is more or less evenly distributed across all traffic rates, which means that the errors are independent of the number of state transitions. It also indicates a good fit of the model to the test set. The

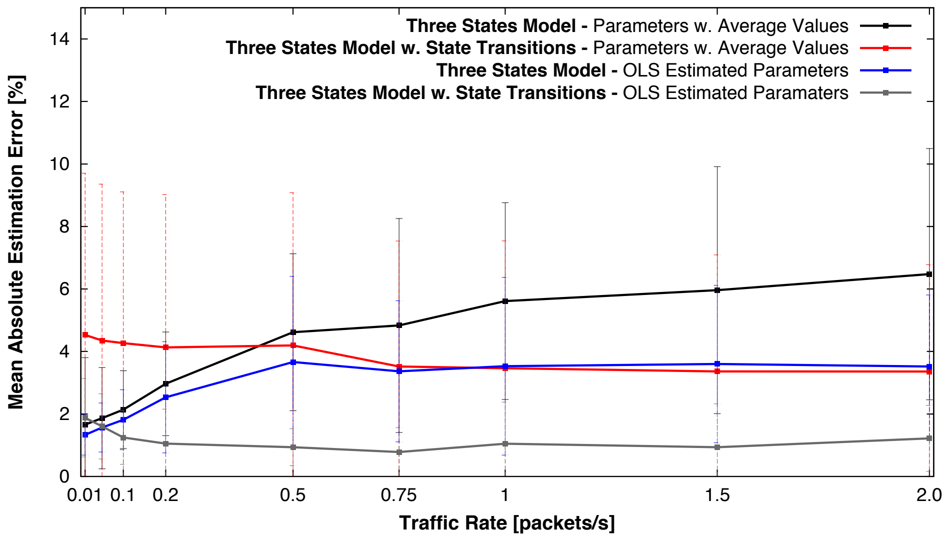

Figure 4.6: MAE vs. Traffic Rate of Per-Node Calibration with Multiple MAC Protocols

errors of the model without state transitions, on the other hand, are small with experiments that feature packet rates below 0.5 packets/s. The degree of the errors of this model increases with an increase of the traffic rate in the experiments. Therefore, the errors of the model are dependent on the number of state transitions of the sensor node, which is low in situations of little network traffic and allows for accurate results.
The mean error of the *three states model* with OLS estimated parameters amounts to 2.77% with a standard deviation of 2.23% and an average R^2 value of 0.9977. The mean error of the *three states model with states transitions* and OLS estimated parameters is 1.19% with a standard deviation of 1.12% and an average R^2 value of 0.9998. The higher R^2 value of the *three states model with state transitions* is an indication that the model is a better fit to the dataset than the *three states model* without state transitions.

4.6.2 Per-Node and Per-MAC Protocol Calibration

The per-node and per-MAC protocol calibration implies that the energy estimation model of a sensor node with a certain MAC protocol prototype was calibrated on the same sensor node with the same MAC protocol prototype. The calibration is performed by conducting a number of power measurement experiments for each sensor node and MAC protocol combination separately. The parameters of the energy estimation model for each sensor node and MAC protocol combination are determined with the measurement results of the identical sensor node and MAC protocol combination. Consequently, this is the most expensive calibration type in terms of time and effort. Figure 4.7 shows the MAE percentages in relation to the traffic rate of the test set used to evaluate the per-node and per-MAC protocol calibration. The training set as well as the the test set are identical to the respective sets used for the evaluation of the per-node calibration with multiple MAC protocols in 4.6.1.

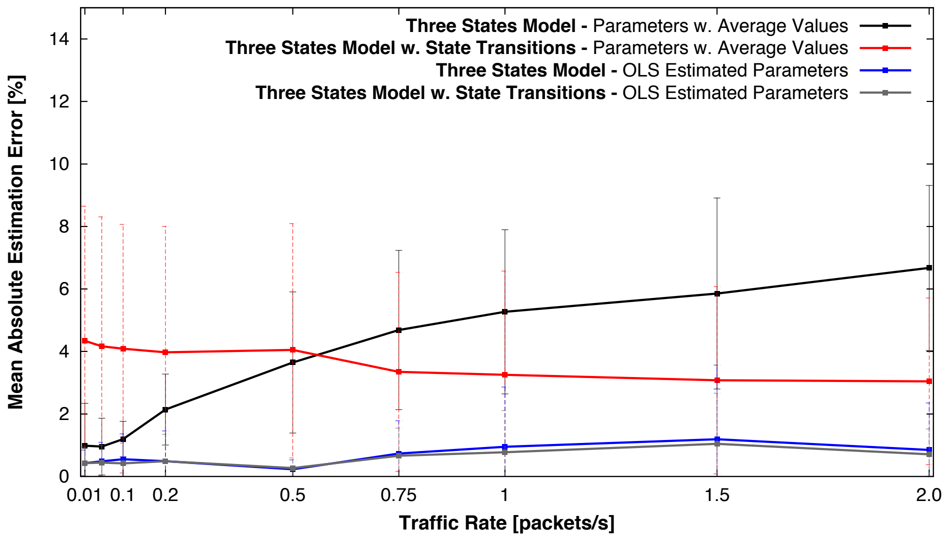

Figure 4.7: MAE vs. Traffic of Per-Node and Per-MAC Protocol Calibration

The mean error calculated across all traffic rates of the *three states model* with average values is 3.49% with a standard deviation of 2.97%. The mean error of the *three states model with state transitions* and average values amounts to 3.71% with a standard deviation of 3.7%.
The mean error of the *three states model* with OLS estimated parameters is 0.66% with a standard deviation of 1.32% and an average R^2 value of 0.9981. The mean error of the *three states model with state transitions* and the same calibration methodology amounts to 0.58% with a standard deviation of 0.95%. The difference of the MAE of both models with OLS estimated parameters is small with an average difference of only 0.08% in favor of the model with state transitions. However, the *three state model with state transitions* features a better fit with an average R^2 value of 0.9999.

4.6.3 Calibration using Different Sensor Nodes and Multiple MAC Protocols

For the last tests we determined the estimation model parameters with experiments using the sensor nodes I, II, III and IV (see Table 4.1) and evaluated the accuracy when using these parameters to calibrate the estimation models on different sensor nodes (V, VI, VII and VIII). Furthermore, power measurements using all of the MAC protocols (CSMA, S-MAC, T-MAC and WiseMAC) were included in the the training set for calibration. In contrast to the previous tests, we only considered the traffic rates 0.05 and 0.5 packets/s for calibration and testing. The training set for determining the parameters of the estimation models consisted of a total of 160 experiments. 10 experiments were conducted per sensor node and MAC protocol combination. The test set with the measurements from the other sensor nodes also consisted of a total of 160 experiments, with 10 experiments per sensor node and MAC protocol combination. Figure 4.8 shows the resulting MAE percentages of each evaluated sensor node/model/calibration methodology combination.
The mean error of the *three states model* with average values calculated across all four sensor

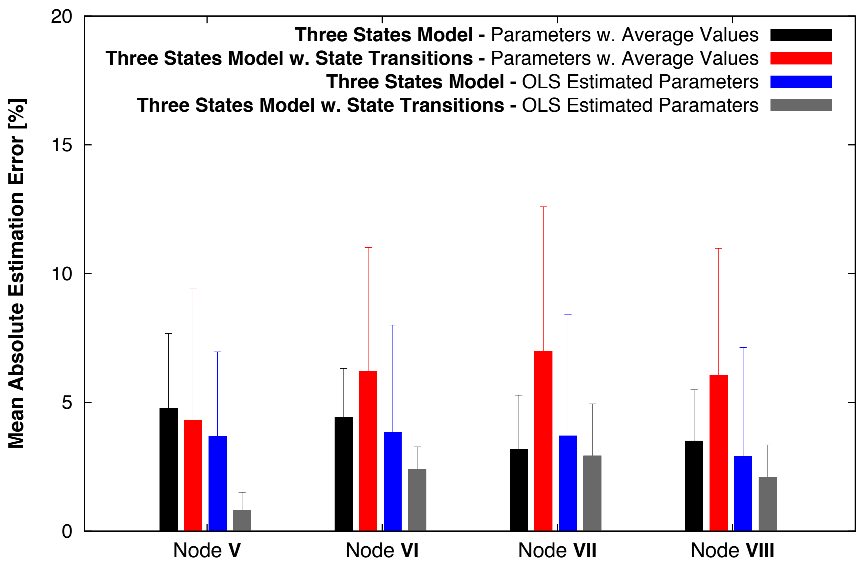

Figure 4.8: MAE of Sensor Nodes when Calibrating the Models with Different Sensor Nodes

nodes amounts to 3.43% with a standard deviation of 2.93%. The mean error of the *three states model with state transitions* calibrated with average values is 6.14% and featured a standard deviation of 5.91%. The reason for the lower accuracy of the model with state transitions is that it performs slightly worse with low traffic rates when calibrated with average values. This behavior can also be observed in the previous test results (Figures 4.6 and 4.8) and occurs at traffic rates below 0.5 packets/s is. However, the resulting mean error across all traffic rates in the other tests are not affected as much, because at high traffic rates the model with state transitions is more accurate than the model without state transitions.

As with the other tests, the accuracy of the models calibrated through regression analysis does not depend on the traffic rate to the same extent as the models calibrated with average values. The mean error of the *three states model* with OLS estimated parameters is 3.11% with a standard deviation of 3.62% and an average R^2 value of 0.9991. The smallest error is featured by the model with state transitions and OLS estimated parameters with a mean error of 2.28% and a standard deviation of 2.08%. Moreover, the model with state transitions also features a better fit than the model without state transitions with an average R^2 value of 0.9996.

4.7 Integrating Sensors in the Energy Estimations

So far, the sensors onboard the WSN nodes were excluded from the energy estimation models. Integrating a sensor for estimating the energy requires minor modifications of the energy estimation models, the addition of a few lines of code to the device driver of the sensor and some adjustments to the methodologies for determining the parameters. In this section, we will demonstrate the addition of a sensor to the energy estimation models with the help of the Sensirion SHT11 sensor [17] which is able to measure the temperature and the relative humidity.

Figure 4.9 shows a graphical representation of the current draw of at sensor node measuring the temperature and subsequently transmitting a packet with the S-MAC protocol. Figure 4.10 shows the same scenario, however measuring the humidity instead of the temperature.

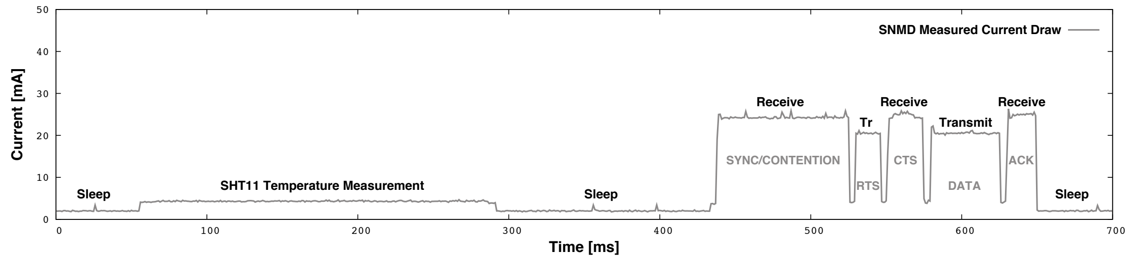

Figure 4.9: Current Draw of a Temperature Measurement and a Subsequent Packet Transmission

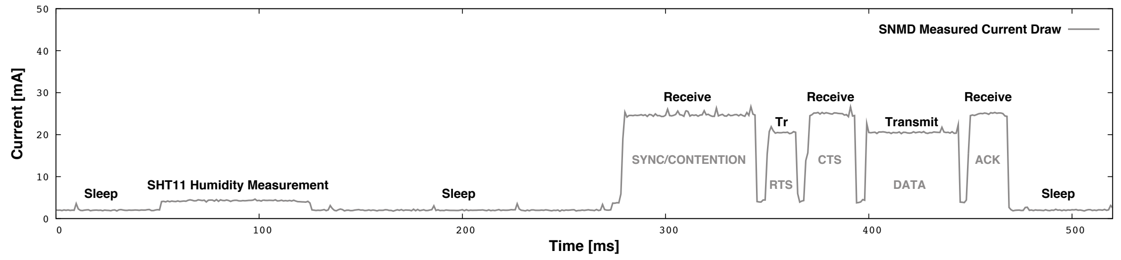

Figure 4.10: Current Draw of a Relative Humidity Measurement and a Subsequent Packet Transmission

The Figures reveal that the SHT11 sensor requires more time to measure the temperature than it does for measuring the humidity. The Figures also reveal that the measurements do not induce the same current draw fluctuation when turning the sensor on and off as does the CC1020 transceiver during its state transitions. Moreover, the sensor node may measure the temperature or humidity in parallel to sleeping, idle listening, receiving data or transmitting data. During a measurement, the current draw I of the sensor will be added to the current draw I of the current sensor node state. For this reason, an active SHT11 sensor does not constitute a sensor node state in its own right. Although we only added one physical sensor to the model, from the viewpoint of the estimation models, we treated the temperature and humidity measurements as two individual sensor entities. This was necessary for accuracy reasons, as the temperature measurements might result in a slightly different current draw than the humidity measurements. The code that has to be added to the SHT11 device driver of the ScatterWeb operating system is identical for both of the energy estimation models. One of the following four function calls has to be added to every position of the device driver code, where the SHT11 sensor is turned on and off for a temperature or a humidity measurement.

```
EnergyEstimation_switchTemperatureSensorOn();
EnergyEstimation_switchTemperatureSensorOff();
EnergyEstimation_switchHumiditySensorOn();
EnergyEstimation_switchHumiditySensorOff();
```

The function calls update the estimation model variables t_{tmp} and t_{hmd}, which represent the respective total durations of the temperature and humidity measurements since initialization of the energy estimation model. Since we do not consider the state transitions of the sensor for the energy estimations, the occurrences of the individual measurements do not have to be counted.

4.7.1 Adding the SHT11 Sensor to the Energy Estimation Models

Apart from the minor modifications of the SHT11 device driver in the operating system, some modifications must also be made to the energy estimation models in order to incorporate the estimations for the temperature and humidity measurements. Figures 4.11 and 4.12 are graphical representations of the estimated current draw of the SNMD measurement shown in Figure 4.10 as modeled with the *three states model* and the *three states model with state transitions*, respectively.

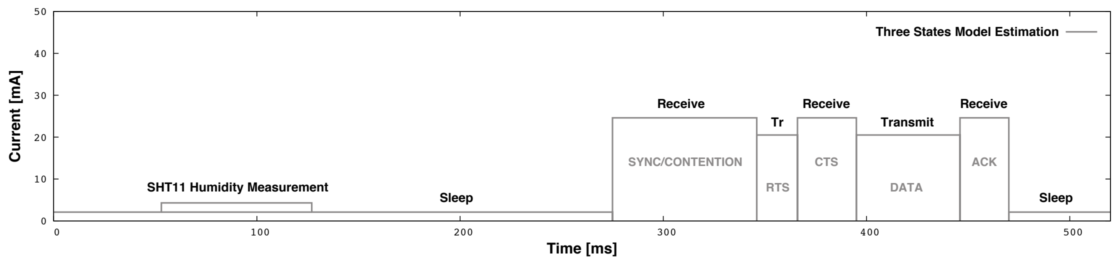

Figure 4.11: Current Draw as Modeled with the Three States Model

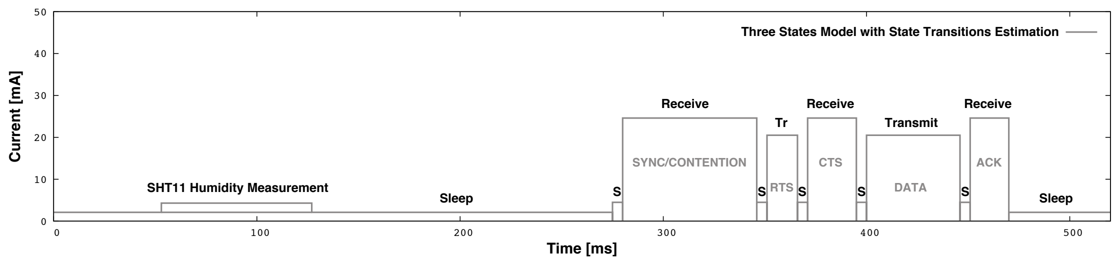

Figure 4.12: Current Draw as Modeled with the Three States Model with State Transitions

To calculate the total power consumption including the SHT11 sensor, the terms $P_{\text{tmp}}t_{\text{tmp}}$ and $P_{\text{hmd}}t_{\text{hmd}}$ have to be added to the energy estimation models, where the parameters P_{tmp} and P_{hmd} denote the raw electrical power of the temperature and humidity measurements. The individual values for the raw electrical power can be derived by multiplying the voltage supply V with the raw current draw I of the SHT11 sensor as shown in Section 4.4. After the energy estimation, the multiplication of the raw electrical power with the measurement durations results in the total raw power consumption of the respective sensor measurements. These values can be added to the power consumption of the sensor node with the transceiver to derive the total power consumption including the SHT11 sensor. Equation 4.14 shows the calculation of the power consumption E with the *three states model* and the added extensions for estimating the power consumption including the SHT11 sensor, where P_{slp} denotes the sleep power, t_{slp} the retention time of the sleep state, P_{rcv} the receive power, t_{rcv} the receive state retention, P_{trns} the transmit power rate and t_{trns} the transmit state retention.

$$E = P_{\mathrm{slp}}t_{\mathrm{slp}} + P_{\mathrm{rcv}}t_{\mathrm{rcv}} + P_{\mathrm{trns}}t_{\mathrm{trns}} + P_{\mathrm{tmp}}t_{\mathrm{tmp}} + P_{\mathrm{hmd}}t_{\mathrm{hmd}} \tag{4.14}$$

Equation 4.15 shows the derivation of the power consumption with the *three states model with state transitions* and the SHT11 extensions, where P_{swt} denotes the power rate and t_{swt} the retention time of the switch state.

$$E = P_{\mathrm{slp}}t_{\mathrm{slp}} + P_{\mathrm{rcv}}t_{\mathrm{rcv}} + P_{\mathrm{trns}}t_{\mathrm{trns}} + P_{\mathrm{swt}}t_{\mathrm{swt}} + P_{\mathrm{tmp}}t_{\mathrm{tmp}} + P_{\mathrm{hmd}}t_{\mathrm{hmd}} \tag{4.15}$$

4.7.2 Parameter Determination of the Energy Estimation Models Including the SHT11 Sensor with Average Values from Power Measurements

In principal, the determination of the model parameters with average values from power measurements is done identically as explained for the *three states model* and the *three states model with state transitions* in 4.4.2 and 4.5.2, respectively. The only difference is the additional establishment of the parameter I_{tmp} (the average current draw of the temperature measurements) and I_{hmd} (the average current draw of the humidity measurements). For the average current draw values, the raw current draw of the SHT11 sensor while measuring the temperature or the humidity has to be determined without the added current draw of other sensor node components like the CPU or the transceiver. An easy way to accomplish this is to make example power measurements with temperature and humidity measurements during the sleep state. Afterwards, the average current draw of the sensor node while sleeping and the average current draws of the sensor node while measuring the temperature and the humidity can be established with the trace files. Finally, the raw average current draws of the sensor measurements can be derived with the difference of the average current draws while measuring the temperature or humidity and the average current draw while sleeping. The establishment of the average values can be done automatically with any number of power measurements. However, possible ranges for the current draw and durations of the measurements have to be determined manually. The ranges for the durations are necessary to clearly distinguish the switch state and the temperature and humidity measurements from each other in the power measurements for the automatic determination of the parameters with a program (see algorithm in pseudo code below). Since both measurement types exhibit a very similar current draw, the only clear way to differentiate them is by their durations. E.g. $I_{\mathrm{hmd}} = 3.52$ mA with a duration of 230 ms and $I_{\mathrm{tmp}} = 3.59$ mA with a duration of 80 ms. We have determined a range of 3 - 5 mA for the current draw of both sensor measurement types, a duration of 200 - 300 ms for the temperature measurements and a duration of 50 - 100 ms for the humidity measurements. The algorithm in pseudo code below illustrates how the parameters are determined automatically from a number of trace files.

The variable i represents the current draw I at time t. The first if-statement assigns value of i to the temporary current draw value of the *sleep* state and increments the occurrence counter s of the same state, if the current draw I is below 3. The next if-statement handles the current draw values between 3 and 5, which is the current draw range of the SHT11 sensor measurements.

The following while-statement counts the number of consecutive current draw values which lie in this range and belong to either the humidity or temperature measurements and stores the value in the *millis* variable. The *millis* variable represents the duration of the current sensor measurement. The next two if-statements add the value of the *millis* variable to a temporary value of the temperature or humidity measurement current draw (I_t, I_h) and increment the respective occurrence counter (t, h), depending on the duration given by the *millis* variable. To derive the average current draw of the measurements, the temporary current draw values of the measurements are divided by their respective counters. Afterwards, the current draw of the *sleep* state is subtracted from these values to derive the raw current draw of the sensor measurements.

```
for all current values $i$ do
  if $i \leq 3$ then
    $I_{\mathrm{s}} \leftarrow I_{\mathrm{s}} + i$
    $s \leftarrow s + 1$
  else if $3 \leq i \leq 5$ then
    $millis \leftarrow 1$
    while $3 \leq (i++) \leq 5$ do
      $millis \leftarrow millis + 1$
    end while
    if $50 \leq millis \leq 100$ then
      $I_{\mathrm{h}} \leftarrow I_{\mathrm{h}} + millis$
      $h \leftarrow h + 1$
    else if $200 \leq millis \leq 300$ then
      $I_{\mathrm{t}} \leftarrow I_{\mathrm{t}} + millis$
      $t \leftarrow t + 1$
    end if
  end if
end for

$I_{\mathrm{sleep}} \leftarrow I_{\mathrm{s}}/s$
$I_{\mathrm{tmp}} \leftarrow I_{\mathrm{t}}/t - I_{\mathrm{sleep}}$
$I_{\mathrm{hmd}} \leftarrow I_{\mathrm{h}}/h - I_{\mathrm{sleep}}$
```

4.7.3 Parameter Determination of the Energy Estimation Models Including the SHT11 Sensor through Regression Analysis of Experiment Results

The basic scheme for establishing the parameters for the energy estimation models including the SHT11 sensor through OLS regression analysis is very similar to the *three states model* and *three states model with state transitions* as described in 4.4.3 and 4.5.3, respectively. The only difference to the described methodologies is the addition of two observed regressors for each model. The regressors are the durations of the temperature measurements t_{tmp} and the durations of the humidity measurements t_{hmd}. A number of physical power measurements and concurrent estimations have to be made in order to obtain the necessary dataset for the estimation of the parameters with the OLS method.

The dataset for determining the parameters for the *three states model* consists of n experiments $\{E_i, t_{\mathrm{slp}_i}, t_{\mathrm{rcv}_i}, t_{\mathrm{trns}_i}, t_{\mathrm{tmp}_i}, t_{\mathrm{hmd}_i}\}_{i=1}^{n}$, where E_i is the observed response (physical measured energy consumption) and the five remaining variables are the observed regressors (obtained with the energy estimation model). Specifically, t_{slp} is the retention time of the sleep state, t_{rcv} is the retention of the receive state, t_{trns} is the retention of the transmit state, t_{tmp} is the duration of the temperature measurements and t_{hmd} is the duration of the humidity measurements. The result of the OLS regression analysis is a set of parameters (β_1, β_2, β_3, β_4, β_5) which can be used to derive the energy consumption E with the following formula.

$$E = \beta_1 t_{\mathrm{slp}} + \beta_2 t_{\mathrm{rcv}} + \beta_3 t_{\mathrm{trns}} + \beta_4 t_{\mathrm{tmp}} + \beta_5 t_{\mathrm{hmd}} \tag{4.16}$$

The parameters for the *three states model with state transitions* are determined with a dataset of n experiments $\{E_i, t_{\mathrm{slp}_i}, t_{\mathrm{rcv}_i}, t_{\mathrm{trns}_i}, t_{\mathrm{tmp}_i}, t_{\mathrm{hmd}_i}, x_{\mathrm{toSlp}_i}, x_{\mathrm{toRcv}_i}, x_{\mathrm{toTrns}_i}\}_{i=1}^{n}$, where x_{toSlp} denotes the number of state transitions to the sleep state, x_{toRcv} the number of transitions to the receive state and x_{toTrns} the number of transitions to the transmit state. The resulting parameters of the OLS regression analysis (β_1, β_2, β_3, β_4, β_5, β_6, β_7, β_8) may be used to derive the energy consumption with the following equation.

$$E = \beta_1 t_{\mathrm{slp}} + \beta_2 t_{\mathrm{rcv}} + \beta_3 t_{\mathrm{trns}} + \beta_4 t_{\mathrm{tmp}} + \beta_5 t_{\mathrm{hmd}} + \beta_6 x_{\mathrm{toSlp}} + \beta_7 x_{\mathrm{toRcv}} + \beta_8 x_{\mathrm{toTrns}} \tag{4.17}$$

A prerequisite for linear regression analysis is a set of observations with linearly independent regressors. The dataset for the *three states model with state transitions* features a higher number of regressors and linear independency is not always given. We have found that the easiest way to achieve linear independency is by conducting a number of experiments which result in different total power consumptions, e.g. by using different traffic rates.

4.7.4 Accuracy of the Energy Estimation Models with SHT11 Temperature and Humidity Measurements

We have evaluated the accuracy of the energy estimation models with a set of experiments that involved temperature and humidity measurements in addition to some network activity. The experiments were very similar to the experiments explained in Section 4.1, where we have used three sensor nodes and a SNMD to concurrently measure and estimate the power consumption of a sensor node. However, we have left out node A (see Figure 4.1) and only used nodes B and C for the experiments. Node B was connected to the SNMD (which physically measured the power consumption of node B), made temperature or humidity measurements and regularly transmitted the value for its estimated power consumption to node C with S-MAC. In the end, the differences of the measured power consumptions to the estimated values were used to derive the mean absolute error (see Section 4.6) of the estimations.

For the evaluation, a total of 20 experiments were conducted with temperature measurements, and 20 experiments were conducted with humidity measurements. In all the experiments, the interval for the temperature and humidity measurements was set to 1 measurement/s and the

interval for transmitting the energy estimations was set to 0.2 packets/s. All of the experiments were made on the same MSB430 sensor node.
The accuracy of the estimations with temperature measurements was determined separately to the estimations with humidity measurements. The training sets for determining the average power values consisted of 5 experiments for each of the measurement types (temperature measurements or humidity measurements). The test set for deriving the accuracy of the models, which were calibrated with average power values, consisted of the remaining 30 experiments (15 temperature and 15 humidity measurements).
The training sets for determining the parameters with the OLS method consisted of 10 experiments for each of the measurement types. To prevent linear dependency of the values of the training set, 5 experiments were added to the training set. The additional experiments were conducted in the same manner as the other experiments, however with different traffic rates and without any SHT11 sensor measurements. The test set for deriving the accuracy of the models, which were calibrated with the OLS method, consisted of the remaining 20 experiments (10 experiments for each sensor measurement). Figure 4.13 shows the resulting percentages of the mean absolute estimation errors.

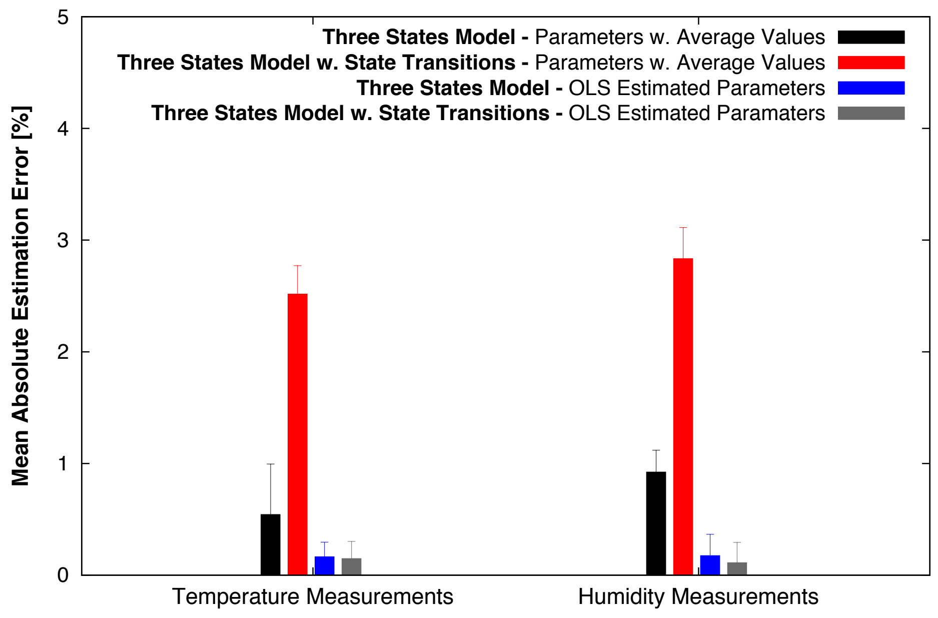

Figure 4.13: MAE of Experiments with Temperature and Humidity Measurements

The mean estimation errors for the temperature and humidity measurements with the *three states model* and average power values are 0.54% with a standard deviation of 0.45% and 0.92% with a standard deviation of 0.19%, respectively. The mean estimation errors of the *three states model with state transitions* calibrated with average values are 2.52% with a standard deviation of 0.25% for the temperature measurements, and 2.83% with a standard deviation of 0.28% for the humidity measurements. We attribute the better result of the *three states model* to the test set which featured a traffic rate of only 0.2 packets/s. The results shown in the Figures 4.6

and 4.8 illustrate that the *three states model* has a higher accuracy with low packet rates (when calibrated with average power values). However, the *three states model with state transitions* easily outperforms the *three states model* with packet rates higher than 0.75 packets/s.
The mean estimation errors of the *three states model* with OLS estimated parameters are 0.17% with a standard deviation of 0.13% for the temperature measurements, and 0.18% with a standard deviation of 0.19% for the humidity measurements. The mean estimation errors of the *three states model with state transitions* and OLS estimated parameters are 0.15% with a standard deviation of 0.15% for the temperature measurements and 0.11% with a standard deviation of 0.18% for the humidity measurements.

Chapter 5

Conclusion

In this thesis we have compared different existing methods for determining the energy consumption of wireless sensor nodes. The energy consumption can be either estimated through a software-based model which is integrated into the operating system of the sensor node or measured with additional onboard hardware components or external generic devices. The advantages and disadvantages of the methods can be summarized with the most prominent characteristics accuracy, cost and ease of use. Measurements with external hardware are the most accurate but require the most expensive deployments in terms of monetary costs and complexity. The methods using onboard hardware are not as accurate but cheaper and less complex to deploy. The cheapest and least complex but also the least accurate methods are estimations using software-based models. Being the cheapest and least complex, however, has a great appeal for use in scenarios outside of a laboratory environment but requires the knowledge of the error rate of such estimations. We have discussed various studies which address this problem and showed that there is a need to systematically quantify the accuracy of software-based energy estimation models for wireless sensor nodes.
We have chosen the MSB430 sensor node platform with the ScatterWeb operating system and four MAC protocols to develop and evaluate two different energy estimation models. The MAC protocols included the energy unconstrained CSMA/CA protocol and the E^2-MAC protocols S-MAC, T-MAC and WiseMAC. The synchronized protocols S-MAC and T-MAC require a precise synchronization methodology. For this we have implemented a unidirectional synchronization scheme and achieved an accuracy of $\sim$1 ms which allows a stable operation of the protocols. A comparison of the energy consumptions of the MAC protocols in relation to the traffic rate showed that the sensor node with the WiseMAC prototype consumed the least amount of energy in all but the highest traffic rates where the power consumption was on par with the S-MAC prototype. The low power consumption comes with the price of a high end-to-end delay which is the shortest for CSMA/CA followed by T-MAC.
To determine the accuracy of the estimation models we have described an experiment setup with three sensor nodes and a SNMD that permitted us to measure and estimate the power consumption of a sensor node in parallel. The comparison of the resulting trace files for the measurements and estimations delivered the error rate of the respective model. We have conducted a wide range of experiments with 8 different sensor nodes, 4 different MAC protocols and 9 different traffic rates amounting to a continuous experiment run time of 130 hours. Part of the experiment data

was used to determine the power consumption deviations among different sensor nodes as this could have an impact on the estimation accuracy. The most conservative results of these experiments showed significant deviations between 0.1% and 4.4% with confidence levels of over 95%.
The first software-based energy estimation model we have implemented was the widespread *three states model*. This model identifies three states to cover all possible sensor node hardware and software configurations in terms of the current draw. Deeper examination of power measurement trace files revealed that during state transitions, the current draw increases or decreases to a certain level for a short period before leveling out to the current draw of the new state. For this reason, we have developed the *three states model with state transitions*, which is able to account for this behavior by counting the number of state transitions during estimation and deriving the energy consumption with a special formula. Both models depend on the predetermination of parameters by analyzing data from a training set. We have described the parameter calibration with average values from power measurements and the calibration through OLS regression analysis of experiment results.
We have evaluated the accuracy of the estimation models depending on the number of sensor nodes and MAC protocols used for calibration and testing. The tests revealed that the average difference across all traffic rates of the two models when calibrated with average values from power measurements was marginal and not significant. However, the error of the *three states model with state transitions* is less dependent on the traffic rate. When we look at the errors of the models at a traffic rate of 2 packets/s, the average error of the three states model ranges from 6.47% to 6.67% while the three states model with state transitions ranges from only 3.04% to 3.36%. This shows that an increase of the traffic rate, which results in increasing occurrences of state transitions, has a significant impact on the estimation accuracy. Therefore, the *three states model with state transitions*, calibrated with average values from power measurements, performs better with high traffic rates and delivers robust and predictable results in cases of varying traffic rates.
The traffic dependency of the models when calibrated with OLS regression analysis is less present, but the accuracy gain of the *three states model with state transitions* is more evident. In all test cases with OLS estimated parameters, the average error of the *three states model with states transitions* is lower than the error of the model without state transitions. The model with state transitions also featured higher R^2 coefficients (see 4.6) throughout all tests, which indicates a better fit of the model to the data. The best results of all the tests were produced by the *three states model with state transitions* and per-node/per-MAC protocol calibration with OLS estimated parameters. It achieved an average error of 0.58% with a standard deviation of 0.95%. The per-node/per-MAC protocol calibration eliminates the problems resulting from power consumption deviations among different sensor nodes and allows a better fit of the parameters to the respective MAC protocol. The estimation of the parameters with OLS regression analysis has the benefit of generating parameters for the energy estimation models which feature a better fit to the datasets than the parameters determined through averaging the values of the power measurements. The drawback of this calibration methodology is that it has to be performed for each node individually with a sufficient amount of power measurements to avoid linear dependency of the training set (see 4.4.3).

The results of the *three states model with state transitions* prove that the higher granularity of the three states model with state transitions has intrinsic advantages as opposed to the model without state transitions. Moreover, the results show that a software-based energy estimation model is an accurate and feasible alternative to determine the energy consumption of wireless sensor nodes where complexity and cost is an issue.
Finally, we have shown that the integration of sensors (e.g. the SHT11 sensor for measuring the temperature and relative humidity) in the energy estimations does not have a negative impact on the estimation accuracy. However, the parameter determination with OLS regression analysis may require a slightly larger dataset because of the additional number of parameters.

Bibliography

[1] SaRonix 32.768 kHz Tubular Crystal NTF3238 / NTF3226 Series Datasheet.

[2] M. Baar, E. Koeppe, A. Liers, and J. Schiller. The ScatterWeb MSB-430 Platform for Wireless Sensor Networks. SICS Contiki Workshop, 2007.

[3] Buettner, M., Gary V. Y., Anderson, E. and Han, R. X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks. ACM SenSys, 2006.

[4] Alan Colvin. CSMA with collision avoidance. *Computer Communications*, 6(5):227–235, October 1983.

[5] N.R. Draper and H. Smith. Applied Regression Analysis, Wiley Series in Probability and Statistics , 1998.

[6] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. Software-based On-line Energy Estimation for Sensor Nodes. IEEE Workshop on Embedded Networked Sensors (EmNets), 2007.

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

[8] Freescale Inc. www.freescale.com.

[9] G Halkes and K. Langendoen. Crankshaft: An Energy-Efficient MAC-Protocol For Dense Wireless Sensor Networks. European Conference on Wireless Sensor Networks (EWSN), 2007.

[10] Haratcherev, I., Halkes, G., Parker, T., Visser, O. and Langendoen, K. PowerBench: a Scalable Testbed Infrastructure for Benchmarking Power Consumption. International Workshop on Sensor Network Engineering (IWSNE), 2008.

[11] Anton Hergenröder, Jens Horneber, Detlev Meier, Patrick Armbruster, and Martina Zitterbart. Distributed Energy Measurements in Wireless Sensor Networks. ACM SenSys, 2009.

[12] P. Hurni, T. Braun, and M. Anwander. Evaluation of wisemac and extensions on wireless sensor nodes. *Springer Telecommunication Systems Journal*, 43(1-2), September 4 2009.

[13] IEEE 802.11 Working Group. http://www.ieee802.org/11.

[14] O. Landsiedel, K. Wehrle, and S. Goetz. Accurate Prediction of Power Consumption in Sensor Networks. IEEE Workshop on Embedded Networked Sensors (EmNets), 2005.

[15] Culler D. Polastre J., Hill J. Versatile Low Power Media Access for Wireless Sensor Networks. ACM SenSys, 2004.

[16] ScatterWeb GmbH. www.scatterweb.de.

[17] Sensirion Inc. www.sensirion.com.

[18] Van Dam T. and Langendoen K. An Adaptive Energy Efficient MAC Protocol for Wireless Sensor Networks (TMAC). ACM SenSys, 2003.

[19] Estrin D. Ye W., Heidemann J. An Energy Efficient MAC Protocol for Wireless Sensor Networks. INFOCOM, 2002.