TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

Traffic aware and Low Power MAC Protocol Analysis and Validation

by
Ing Anton Bilos

Supervisor:

Prof. Dr. Johan J. Lukkien

Dr. Ir. Richard P.H.F.M Verhoeven

November 12, 2009

Abstract

Wireless sensor networks are becoming more and more commonplace and is also the focus of a lot of research in improving these networks. As the wireless sensor hardware has limited memory and energy resources most of the research is in improving the efficiency of the software running on the sensors.

The radio and processor have features to reduce power consumption like sleeping modes while they are inactive. To reduce power consumed by the radio, the protocol needs to exploit these power saving features while keeping in mind the communication requirements. The same holds for the processor, the processor needs to sleep as much as possible but needs to respond quickly to tasks. These conflicting requirements present a problem in the practical implementation of wireless sensor networks. In this thesis we explore the protocol TRAWMAC and the operating system OsasOS to improve power efficiency over the current system in use.

Another strategy in conserving power is adjusting the software on the amount of traffic being relayed and the speed at which the traffic needs to be transmitted. This improves power consumption under a wide range of applications.

To measure the power consumption that we have saved, we have chosen to directly measure power consumption via specialized hardware.

Next to improving power consumption of a wireless sensor it is also important to reduce processor resource usage like the ROM and RAM memories. Saving memory usage allows larger and more complex applications to be used on the same platform or makes it possible to use processors that contain less ROM and RAM memory.

Preface

I started my master study in embedded systems at the Technical University of Eindhoven in 2005 after completing my HBO bachelor in computer science. During this period I followed the course on wireless sensor networks given by Johan Lukkien. He showed that the field in wireless sensor networks is very diverse with many applications and many open problems. The broadness to this field and the strong link with efficient small embedded systems appealed to me.

Johan Lukkien offered me an interesting project that centered around improving the energy performance of the wireless sensors. This project seemed to be an interesting project to improve the current wireless sensor network system. It provided a broad scope of problems and solutions combining with my interest in low level systems.

I would like to thank Johan Lukkien for providing me with an interesting master project and his insightful feedback during the project and on this thesis. I would also thank Richard Verhoeven and Remi Bosman for providing day to day support and feedback on smaller and larger issues. I would also like to thank all the people in the SAN group for their friendship and I will especially miss the conversations during lunches and in the hallways. Special thanks go out to the occupants of office 5.04 for the exhilarating speed chess matches.

Contents

Preface **3**

Contents **7**

List of figures **10**

List of tables **11**

Acronyms **13**

1 Wireless sensor networks introduction **15**
1.1 Wireless sensor networks . . . 15
1.1.1 WSN applications . . . 15
1.1.2 WSN structure . . . 16
1.1.3 MAC protocol . . . 21
1.1.4 Application . . . 23
1.2 Constraints . . . 23
1.2.1 Microcontroller resources . . . 24
1.2.2 Power . . . 24
1.3 Thesis work . . . 25
1.3.1 Motivation . . . 25
1.3.2 Research questions . . . 25
1.3.3 Thesis focus . . . 25

2 MAC protocol **27**
2.1 Taxonomy of MAC protocols . . . 27
2.2 Introduction . . . 29
2.2.1 Protocol features . . . 29
2.2.2 Design choice . . . 30
2.3 TRAWMAC . . . 30
2.3.1 Preamble sampling . . . 30
2.3.2 Implementation . . . 32
2.3.3 Neighborhood detection . . . 34
2.3.4 Traffic awareness . . . 35
2.4 Problems faced . . . 36
2.5 Missing functions . . . 36

3 MAC Programmer Interface **39**
3.1 Introduction . . . 39
3.2 Related work . . . 39
3.2.1 Advantages . . . 39
3.3 Traffic patterns . . . 40
3.3.1 Traffic patterns and applications . . . 40
3.3.2 Pattern requirements . . . 41
3.3.3 Network state information . . . 42
3.4 API . . . 42
3.4.1 Function classes . . . 43
3.4.2 Data structures . . . 43
3.4.3 Calls . . . 44
3.4.4 Functionality . . . 45
3.5 Transmit and Receive . . . 45
3.6 Settings . . . 46
3.7 Neighborhood . . . 46

4 OsasOS operating system **49**
4.1 Related work . . . 49
4.2 Introduction . . . 49
4.3 Scheduler . . . 49
4.3.1 Scheduling API . . . 50
4.3.2 scheduler operation . . . 51
4.3.3 Example application, blink . . . 52
4.4 Drivers . . . 54
4.4.1 Example driver ADC . . . 54
4.4.2 Radio . . . 54
4.4.3 Microcontroller . . . 56
4.4.4 Peripherals . . . 56

5 Experiments **57**
5.1 setup . . . 57
5.1.1 Resource measurement . . . 57
5.1.2 Energy measurement . . . 57
5.2 Measurements . . . 60
5.2.1 WSN hardware . . . 60
5.2.2 operational measurements . . . 61

6 Results **63**
6.1 Baseline . . . 63
6.2 WSN hardware . . . 63
6.3 operational measurements . . . 64
6.4 TRAWMAC detailed . . . 65
6.5 WSN stacks . . . 65
6.6 Resource usage . . . 66
6.7 Energy model . . . 66
6.8 Conclusions . . . 67

6.9 Improvements 68
6.10 Future work 69

Bibliography **70**

List of Figures

1.1 *Usage of wireless sensor network in monitoring of cows for the WASP project [con09]* 16

1.2 *Abstract view of hardware buildup and actual physical hardware.* 17

1.3 *Layers within the wireless sensor network stack. At the bottom are the operating system and hardware drivers dealing with the node functionality directly by manipulating the I/O registers and having interrupt handlers. The MAC protocol uses the radio driver to communicate. The network layer uses the MAC in turn to provide multi hop communication to the applications above the network layer. The applications use some driver functionality to read out sensors and the operating system for task scheduling.* 19

1.4 *Node link diagram showing the physical makeup of a wireless sensor network. The links between the nodes signify that these nodes are in radio range of eachother. The boxes with the gateway label provide an interface where the outside world can communicate with a wireless sensor network. Nodes that have a dotted lining are sensing nodes while the nodes with a solid line are routing nodes.* 21

1.5 *Taxonomy of various MAC protocols adapted from the MAC chapter by Langendoen et al [Lan07]. In this list the four main classes of MAC protocols is shown with the corresponding protocols. The protocols are organized by publication date and relationship.* 22

1.6 *In this figure we show the changes made in the software stack of the wireless sensor network. The gray parts are where the changes are made and where new layers are inserted.* 26

2.1 *Taxonomy of various MAC protocols adapted from the MAC chapter by Langendoen et al [Lan07]. In this list the four main classes of MAC protocols is shown with the corresponding protocols. The protocols are organized by publication date and relationship.* 27

2.2 *Micro Frame Preamble scheme of TRAWMAC* 31

2.3 *Data Frame Preamble scheme of TRAWMAC* 31

2.4 *Micro Frame Preamble with acknowledgements* 32

2.5 *This state machine shows how TRAWMAC handles reception of data addressed to this node. The reception sequence started at a detection of a MFP addressed to the current node. At any place in the reception states a timeout can occur so the stack returns into a valid state for the next CI. The timeout duration is set to a small constant value.* 33

2.6 *This statemachine shows how TRAWMAC handles transmission of data. The transmission sequence is started when a packet is pending in the transmit queue. At any place in the receive section a timeout can occur so the stack can return to a state where it can restart operation.* 33
2.7 *New nodes and other nodes that receive surrounding chatter from other nodes use the information that is in each MFP and ACK to populate and renew their neighborhood table.* 34
2.8 *Check Interval (CI) adjustment in TRAWMAC happens always at a power of two times the current CI. This creates extra time slots while keeping in synchronization with the other nodes that have not updated to the new CI.* . . 35

3.1 *Influence of duty cycle on the packet latency. The shorter the duty cycle the shorter the time that the packet is offered for transmission and and actually transmitted.* 41
3.2 *If we assume that each transmission slot allows only two packets to be transmitted, then the amount of slots limit the amount of packets that can be received or transmitted.* 42
3.3 *These sequence diagrams show how transmission and reception performed by the application and how the MAC protocol handles this. The transmission diagram also shows how the MAC retrieves a packet from the transmission queue but that the MAC API computes the new deadline value for this packet just before transmission.* 46
3.4 *This sequence diagram shows how the MAC settings setup is performed. The values from the call are translated to MAC specific values and then loaded through the interface functions that the MAC provides, this can differ on a MAC by MAC basis.* 47

4.1 *Scheduling within OsasOS. Tasks are generated by the periodic task list and some sporadic tasks are generated by interrupts. These pending tasks are placed in a pending task FIFO queue. Interrupts (the periodic task list is also interrupt driven) wake up the CPU and runs the task execution loop. This loop checks the queue for any pending tasks and then executes these. When the queue is empty, the processor is put back to sleep.* 50

5.1 *Schematic of the location of the energy measurement device. The device itself sits in between the user of power and a battery pack. The current flowing trough the device is measured over time.* 58
5.2 *Circuit board layout of the analog section. Contains the amplifier, reference, ADC and 3.3V regulator for the wireless sensor.* 59
5.3 *Digital section with display and buttons* 60

6.1 *Power consumption graph, using the values of table 6.4* 65

List of Tables

1.1 *Microcontroller sleep modes. Most microcontrollers have various sleep modes, they vary in naming but most of them adhere to this pattern. Some controllers have additional modes providing more fine grained control of power consumption.* 17

1.2 *Power consumption statistics of various devices on the BSN [Lon] nodes. The device power consumption information is taken from the datasheet of the CC2420 [Ins07] and the MSP430F149 [Ins04] microcontroller.* 24

2.1 *MFP, Acknowledgements and Beacon structure* 34

3.1 *Traffic patterns in wireless sensor networks classified by latency tolerance, throughput and periodic traffic.* 40

3.2 *Packet structure* 43

3.3 *Settings structure* 43

3.4 *Neighbor structure* 44

3.5 *Possible Error codes from the API functions.* 44

4.1 *Task structure* 50

6.1 *Calibration measurements of the energy measurement device. The low accuracy at the lowest level is due to lack of resolution from the measurement device.* . 63

6.2 *Power consumption of wireless sensor peripherals.* 64

6.3 *Power consumption of some operations like those from the MAC layer and sensor readings* 64

6.4 *TRAWMAC operational measurements, first number is the transmission power consumption, second number is reception power consumption. The values are in micro ampere's and are measured with a 3.3V power supply at temperature of 20 degrees centigrade.* 65

6.5 *Power consumption of other wireless sensor network stacks.* 66

6.6 *Comparing resource usage between the OsasOS and TRAWMAC stack with MantisOS and TinyOS* 66

6.7 *Power consumption figures of each radio mode used by the TRAWMAC energy model. These values are taken from the datasheet [Ins07] and assume an supply voltage of 3.3V.* 66

6.8 *Time durations of various TRAWMAC actions.* 67

Acronyms

General acronyms

WSN	Wireless Sensor Network
MAC	Media Acces Control
CSMA	Carrier Sense Multiple Access
TDMA	Time Division Multiple Access
CMOS	Complementary Metal Oxide Semiconductor
RTS	Request To Send
CTS	Clear To Send
ISR	Interrupt service routine
MCU	Micro Controller Unit
CI	Check Interval
CCA	Clear Channel Assessment
ADC	Analog to Digital Converter
SPI	Serial Peripheral Interface
BER	Bit Error Rate

Terms

Node	One participant in a wireless sensor network.
Mote	Physical hardware participating in a wireless sensor network.
Tick	Time unit used by OsasOS, it is 1024th of one second.
Preamble	Small packet sent by a transmitter to signify that it wants to communicate.
Idle listening	Time that the radio receiver is active but no relevant data is received.

Chapter 1

Wireless sensor networks introduction

Before we start with the actual subject of this thesis we need to describe the context in which it is performed. We need some advance information on how wireless sensors are built up and what problems they face. This makes it a easier to understand the work that is presented in this thesis.

1.1 Wireless sensor networks

A Wireless Sensor Network (WSN) is a network comprised of small computing platforms equipped with sensors that communicate with an entity that reads out the sensors via a wireless link. One of these individual computing platforms is also called a **node**. The entity that reads out sensor information of the network is called a **gateway**. The entities in the wireless sensor network perform four principal functions. Sensing of values in the networks, Actuation of actuators in the network, communication of values to and from the actuators and sensors and computation in the network performed on the values.

1.1.1 WSN applications

Wireless sensor networks are used in applications where monitoring would be difficult or expensive to do. The nodes in a wireless sensor network are scattered in the environment that needs to be sensed. The environments can be varied, ranging from large forests in forest fire detection [HB07], personal medical monitoring [AW06] to infrastructure monitoring [NX04]. Most of the nodes are placed in the environment without any easy way to replace them. In the case of the forest fire detection, the nodes can be scattered around in a forest from an airplane. When monitoring a structure the nodes can be embedded into the structure itself. In both of these cases, replacing the node or changing the battery is time consuming, expensive or sometimes not possible.

The work done in this thesis is done in support for monitoring of livestock (under the WASP project [con09]) and in a medical setting.

In our use case livestock monitoring is performed by attaching one node with an accelerometer sensor to the legs of a cow. The information coming from this sensor when the cow is

(a) cow with sensor **(b)** leg sensor detail

Figure 1.1: *Usage of wireless sensor network in monitoring of cows for the WASP project [con09]*

walking is sent to the gateway. This information is analyzed to determine the condition of the cow, if it is walking correctly, how much it is walking and what other activities it is doing.

In the medical sensing use case we attach the same nodes from the livestock monitoring to patients. The accelerometer can determine if a person is standing up, walking or running. Other sensors can also be used like temperature, heart rate sensing to determine patient health or what activities the patient is doing. This information can be used by doctors to determine patient health or when some exceptional conditions occur where medical intervention is required (fall, heart attack).

1.1.2 WSN structure

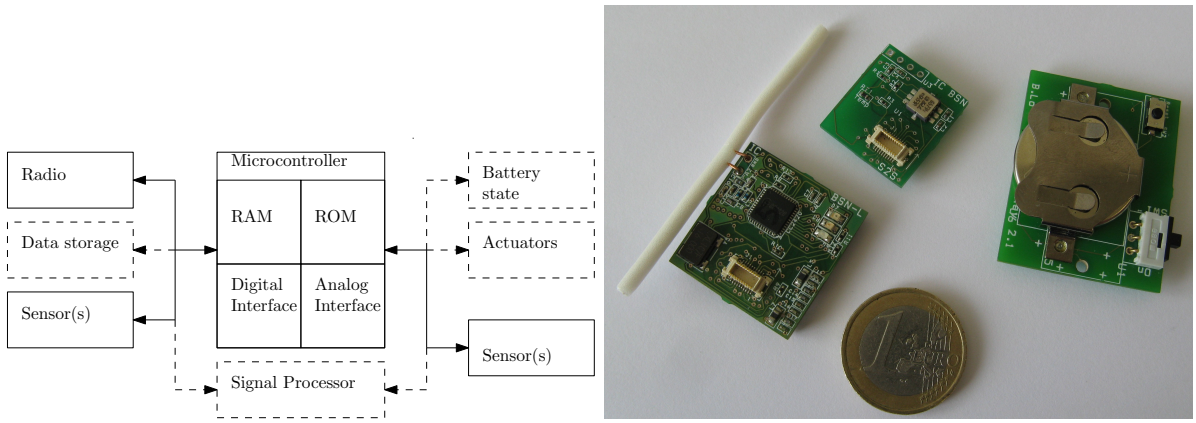
How are wireless sensor networks structured? We will go into the physical structure of wireless sensor networks, how the nodes interact with each other. But also how the software and hardware is internally structured.

WSN node hardware

The hardware and software in a WSN node are structured in a particular way.

The hardware of a wireless sensor node is made up out of three essential components: central processing unit, sensors, radio. The central processing unit consists usually of a microcontroller with limited resources in permanent storage (usually Flash ROM memory) and working memory (RAM memory). The microcontroller contains the wireless sensor software stack and hardware drivers to control the radio and the sensors. The controller has some integrated units, like serial interfaces, Analog to digital converter, digital to analog converter, timers and more. Disabling these units individually lowers power consumption. These units are usually called peripherals in literature.

The microcontroller has various operational modes. The difference between these modes is what functional units of the controller are operational and how much power it consumes. The active mode of the controller is the mode where instructions from program memory are executed, all the other modes disable the processor but can wake up the CPU to handle events or resume execution. The microcontroller power modes progressively consume less and less



(a) Hardware schematic **(b)** Actual hardware

Figure 1.2: *Abstract view of hardware buildup and actual physical hardware.*

Mode	peripherals	Clocks	Wake up
Active	CPU and the rest	All	All
Idle	CPU off, rest on	All	All
Stand by	CPU off, ADC off, timers off	I/O clocks on	Only active peripherals
Sleep	Interrupt, or other digital	All clocks off	Only external interrupts or digital peripherals

Table 1.1: *Microcontroller sleep modes. Most microcontrollers have various sleep modes, they vary in naming but most of them adhere to this pattern. Some controllers have additional modes providing more fine grained control of power consumption.*

power while reducing the amount of sources the CPU can wake up from and switch to active mode. A few microcontrollers have been reviewed and have more or less the same four power saving modes, see table 1.1. Some vendors have more power modes that provide more fine grained control or extend one mode with some extra flexibility.

The radio chip is usually an integrated low cost transceiver working on an ISM frequency band. They are usually connected to a serial interface to setup frequency bands, parameters and writing/reading the transmit/receive frame queues. The radio's usually have various operational modes like transmitting, receiving, idle and shutdown modes. The difference between the idle and shutdown mode is that in idle mode the frequency synthesizer and crystal are operational, this way the radio can quickly switch to receiving or transmitting mode. The shutdown mode powers down all clocking circuitry reducing power consumption almost to zero, the disadvantage of this is that startup to idle mode (and hence receiving and transmitting) takes longer then switching from idle to receiving or transmitting. The time difference between these two modes varies between radio chips but is on the order of a few milliseconds.

The sensors measure the environment and transform these values into electrical signals the node processor can read out and process. There are a lot of physical quantities that can be detected with sensors, for example: temperature, acceleration, pressure, light level, airborne chemicals and many more.

These sensors present electrical signals that represent the quantity of the relevant physical unit that is sensed. The value of the electrical signal is expressed in a voltage. There are two modes of representation of the values to the microcontroller. This can be done with analog signals or digital signals. In analog signals the physical quantity is represented by a varying voltage that expresses the measured quantity. This voltage is read out by the microcontroller via an analog to digital (ADC) converter that translates the voltage level to a digital value. The accuracy of this ADC determines the resolution of the voltage to be sensed. Another mode of sensor operation is digital, the sensor outputs an logical one or zero. These logical values are usually expressed as maximum supply voltage for one, and no voltage for zero. There is another variation on this scheme called PWM (Pulse Width modulation). In this scheme pulses of ones and zero levels are given, the length of the high level (or low level) encodes the measured value.

Electrically there are some differences between analog or digital sensing. For sensing an analog value, you need an ADC but also circuits to filter and condition the signal. Digital signals are a lot simpler and are more immune to interference from the outside world. Another property of analog sensing is that it is more power intensive then digital. The reason for this that the switching elements like transistor are placed in their linear region, behaving like controllable resistors consuming power. With digital there is no linear operation, transistors are fully off or fully on (having very little resistance).

Most analog sensors do not have any power management functions like the controller or radio chip, so other methods of power control is required. Usually the power is switched on and off to the sensor array and a small settling time is needed. Some sensors have power management functions to reduce power consumption to a minimum when not used.

Some nodes have some extra hardware attached that makes it possible to communicate with a larger system like a PC or in general an internet connected device, these nodes are also called gateway nodes. The gateway node exposes the functionality of the wireless sensor network to the internet. This way the user can remotely measure the values of the wireless sensor network.

WSN node software

The software that is running on wireless sensor network nodes are usually represented in a layered stack that has the strucure of an application with underlying communications stack and an operating system giving hardware control, see figure 1.3.

The operating system controls the hardware of the node with hardware drivers and provides task scheduling functions for the applications and drivers. The communications stack is also divided up into layers that increase in abstraction level of the wireless sensor network. The layers are reminiscent of the OSI network stack model. There is a physical interface layer, Media Access Control (MAC) layer, routing layer, transport layer and application layer.

The main control of the resources of the wireless sensor node is provided by the operating system. The operating system provides drivers to control the resources of the microcontroller like timers, analog to digital converter and serial busses. The operating system also provides a scheduler for tasks the application, drivers and networking stack want to execute. There are various operating systems in existence that have been made for low resource systems (Contiki [DGV04a]) or specific to wireless sensor nodes (TinyOS [PLC03], MantisOS [SBH05])

The networking stack consists of a few layers (hence a networking stack) that provide communication functionality to the network. The lowest level layer is the physical layer that

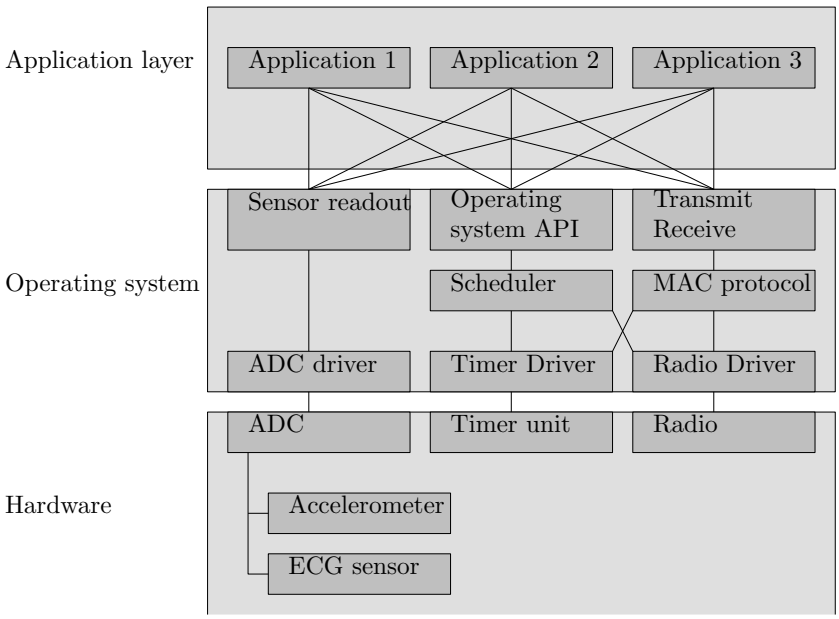


Figure 1.3: *Layers within the wireless sensor network stack. At the bottom are the operating system and hardware drivers dealing with the node functionality directly by manipulating the I/O registers and having interrupt handlers. The MAC protocol uses the radio driver to communicate. The network layer uses the MAC in turn to provide multi hop communication to the applications above the network layer. The applications use some driver functionality to read out sensors and the operating system for task scheduling.*

interfaces with the radio chip. Usually it is part of the operating system and is made specially for the radio chip used. The MAC (Media Access Control) layer uses the radio chip driver to provide communication functions to the direct neighbors of the node. The MAC layer can be one of many protocols in existence, examples of these are S-MAC [YHE02], TRAMA [VR03], WISEMAC [EHD04] B-MAC [JP04] and many more. The MAC layer is subject of much research, so there are a lot of MAC protocols in existence.

To provide communication between nodes that are not direct neighbors, a network/routing layer is needed. This layer makes communication possible through the whole network. To provide communications over multiple nodes in a path (multi hop) a router needs to be incorporated that directs traffic in the right direction. There are many different routing layers like TORA [ND99] and many more [KA03].

To provide reliable links, a transport layer is added on top of the network layer. This layer handles reliable point to point communication, hiding the network layer details and providing point to point channels. This layer is often encountered in general network communication like the internet but is not always present on wireless sensor nodes. There are several reasons for this. The communications medium in wireless sensor networks has a higher BER (Bit Error Rate) and smaller frame size, this complicates the implementation of an efficient transport protocol. Also, many applications do not need an transport layer. It is also costly in terms of power and resources.

On top of the whole networking stack and operating system runs the application. The application layer defines the behavior of the node and also the network itself. Most wire-

less sensor network applications consist of reading out a sensor value and relaying this to an interested gateway. Some enhancements to this scheme can be made by using filtering, aggregation and on node processing.

There has been a lot of research in application frameworks that run on top of the network and provide an application framework where applications can run on. The programming model provides a abstraction of the wireless sensor network and its nodes and makes it possible to update and change the network behavior over the air. The TinyDB [MFHH05] framework provides an SQL like abstraction language to query nodes for sensor values. The nodes themselves run the TinyDB application on nodes running TinyOS operating system. The users can construct SQL queries that are distributed on the wireless sensor network and retrieve data as results from such a query. The OSAS [BLV09] framework provides a virtual machine platform for programming wireless sensor networks. The framework consists of a virtual machine running on the wireless sensor nodes, a compiler toolchain and a simulator. The nodes are programmed over the air and can be dynamically reprogrammed during the lifetime of the network.

current software stack

Now that we know what hardware and software is used in wireless sensor networks, let us describe the system used in this thesis.

The nodes used are the BSN nodes [Lon] (Body Sensor Network) developed by Imperial College London. There are two versions of the wireless sensor nodes, initially version one was used, version two is more compact and has a larger microcontroller.

The hardware of the nodes is built up out of a main board that contains the controller, radio and serial flash chip. To this board sensor boards can be connected, this arrangement makes the platform suitable for various sensing applications.

The microcontroller used is the Texas Instruments MSP430F1611 [Ins04]. The microcontroller has 48KiB Flash memory for its program and 10KiB of RAM. The microcontroller has a 12Bit ADC for sampling analoge signals, two 16 bit timers with multiple capture compare units, external interrupts and two serial busses that can be configured to SPI, I2C or UART interfaces. The microcontroller is clocked by an internal RC (Resistor Capacitor) oscillator and a separate oscillator clocked by an 32.768Hz crystal.

The radio used is a Texas instruments CC2420 [Ins07] transceiver compatible with the IEEE 802.15.4 standard. The transceiver has a transfer rate of 250KiBits per second and operates at the 2.4GHz ISM band. The radio uses a SPI bus and several I/O's and registers for control, it has on-board FIFO buffers, encryption engine and protocol support for the IEEE standard.

The main board with the controller and radio also contains an AT45DB041D [Cor09a] serial flash chip with a capacity of 512KiBytes. The flash chip can be used for temporary storage or logging of data, some applications use it as a store for firmware updates.

The sensor boards attached to the main node determine the sensing capabilities of the node itself. There are various sensor boards available like acceleration sensors and ECG monitor. The sensors can be turned on and off under control of the microcontroller, this is done to conserve power when the sensors are not in use.

The BSN nodes are based on the common Telos-B [Tec] "standard" of nodes and behave as such. The TinyOS wireless sensor network operating system has been ported to these nodes. The operating system uses a 802.15.4 physical layer for communication with no real

MAC to speak of.

WSN network

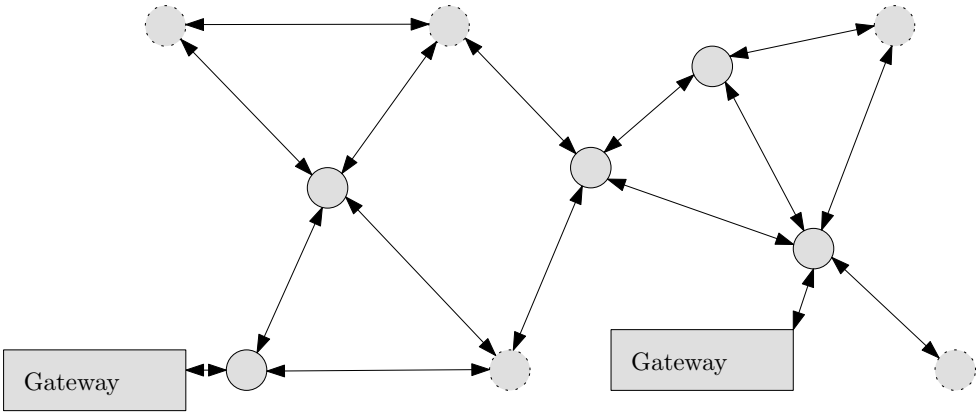


Figure 1.4: *Node link diagram showing the physical makeup of a wireless sensor network. The links between the nodes signify that these nodes are in radio range of eachother. The boxes with the gateway label provide an interface where the outside world can communicate with a wireless sensor network. Nodes that have a dotted lining are sensing nodes while the nodes with a solid line are routing nodes.*

Figure 1.4 shows a abstracted view of one example wireless sensor network. There are a group of nodes that all have a radio to relay their sensor values to the gateway. As the radio has limited range only the nodes near the gateway can communicate directly with it. There may be multiple gateways in the network and multiple sensors.

Communication between neighbors is achieved via the use of a MAC protocol that uses a shared medium. Manages access to this shared medium and provides semantics for communication to its neighbors. Usually the nodes in a wireless sensor network form a mesh network but as the nodes have limited range, not all nodes can directly communicate with other nodes. This problem calls for the usage of a network layer abstraction that is able to route data from one node to another over other nodes.

Wireless sensor networks often contain nodes that are mobile, This means that the structure of the network changes over time. This introduces problems for the MAC and network layer. Usually these two layers record information about the topology, the MAC for its direct neighbors and the network layer for the overall structure. As nodes move around, the neighbors the node can reach change and also the topology changes. The MAC and network layer need to be robust to changes if deployed in such an environment but this proves to be hard and energy consuming.

1.1.3 MAC protocol

We have briefly touched on the MAC layer in the treatment of wireless sensor network software architecture in chapter 1.1.2. But what is a MAC protocol and how does it influence the wireless sensor network?

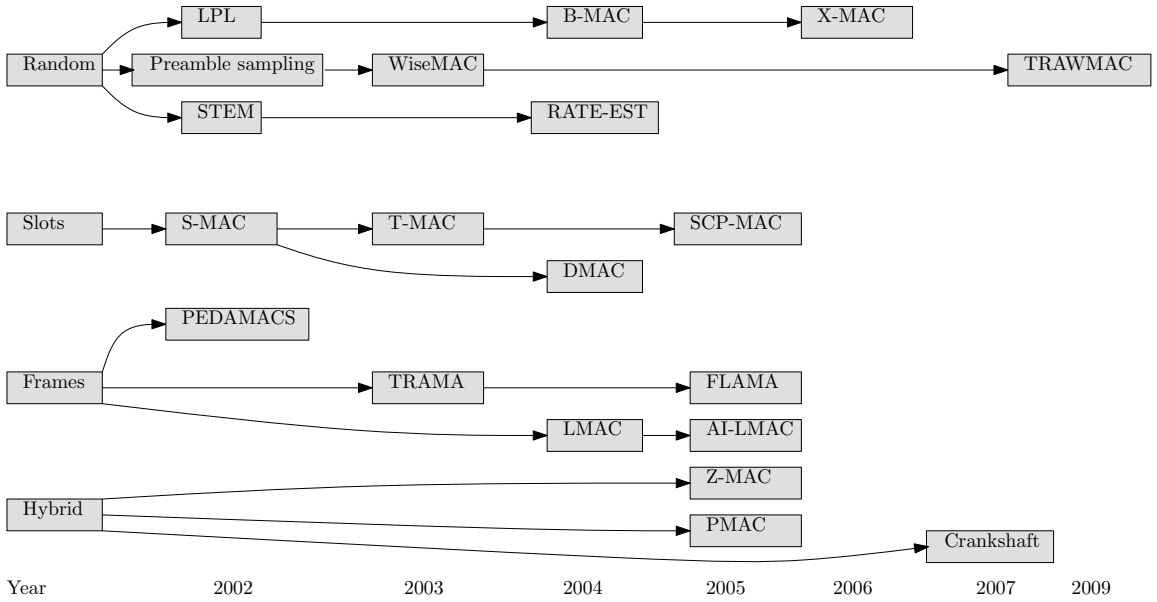


Figure 1.5: *Taxonomy of various MAC protocols adapted from the MAC chapter by Langendoen et al [Lan07]. In this list the four main classes of MAC protocols is shown with the corresponding protocols. The protocols are organized by publication date and relationship.*

The MAC layer/protocol is the communications protocol that makes it possible to communicate with direct neighbors. The protocol defines what actions are performed during communication and what format the communications will have. Communication is performed by using data elements called *frames*, these are exchanged between nodes.

The MAC has a large influence on extra-functional characteristics of the wireless sensor network as it regulates these characteristics. These characteristics include power consumption, measurement latency and data throughput. Usually the power consumption is the most important aspect, but sometimes a tradeoff is made with latency and throughput.

There is a large variety of MAC protocols in existence, comprising a veritable "alphabet soup" of protocols. Langendoen et al has created a comprehensive paper [Lan07] describing many protocols. The MAC protocols are split up into three classes of protocols differing in how time on the medium is partitioned.

Random Random period protocols, wake up and sleep protocols at random periodic intervals. The periods and intervals is randomly filled it but does not require any prior synchronization.

Slots In slotted protocols, the nodes wake up and sleep at the same time. All nodes wake up at the same time and contend for the medium for a period. The simultaneous wake-up requires some synchronization.

Frames In framed protocols, time is divided into *frames* that is distributed across all nodes. All nodes occupy one or more slots and create a conflict free assignment. This is the most complicated assignment of all three classes.

hybrid Hybrid protocols borrow one or more concepts from the three other classes to counteract their deficiencies.

1.1.4 Application

The application runs on top of the communication stack and the operating system and provides the functionality of the wireless sensor network.

Wireless sensor networks are used in a whole range of applications that can be subdivided in a few classes. There are long lifetime monitoring applications where network lifetime is the most important aspect. These applications are usually monitoring of large objects where the replacement and upgrading of nodes is hard or impossible. Examples of such applications are forests [HB07] and infrastructure [NX04]. Another class of applications is where node communications is important, the communication needs to be fast and or reliable. These communication requirements conflict with power efficiency so power efficiency is usually not primary in such applications. These nodes are also usually more accessible so replacement of batteries or nodes is not such of a issue, still a long lifetime is desirable in such applications. Example of these kind of applications is medical monitoring [AW06].

Most simple wireless sensor network applications are nothing more then programs that read out a sensor each time period and transmit through the radio. Many applications do not require much more complexity.

This does create a few problems, what if the user wants to read out the sensor more quickly? What if the sensor is noisy and some more processing is required? What if the average of all sensor values is needed? Or the maximum? A lot of these functions can be implemented on the user side but at considerable communication overhead. Some functionality is fixed in the node and requires reprogramming.

To improve flexibility there are various schemes in remote programming of wireless sensor nodes. One way to accomplish this is by reloading the entire firmware of the wireless sensor node. The firmware is sent through the network and the nodes reprogrammed, these reprogrammed nodes in turn forward their updated firmware through the network.

The scheme of reprogramming the entire firmware of a node has a few disadvantages. The microcontroller of nodes do not have address translation or other mechanisms of application isolation so the entire firmware needs to be reprogrammed. Partial updates requires OS specific functions and special compiler directives. Another way for in-network reprogramming of nodes is through a virtual machine or runtime that executes scripts or byte code. Examples of these systems are Kairos [RGG05], MATE [LC02], TinyDB [MFHH05] and OSAS [BLV09]. These application frameworks provide methods for in network reprogramming and also partial reprogramming. The control that these frameworks have make it possible to add new functionality to the network without disturbing existing applications and also make it possible to provide advanced functionality like aggregation and on node processing.

1.2 Constraints

Wireless sensor networks face a few design constraints in their implementation. As these sensors are usually embedded in the environment and have their own energy supply (batteries) and optionally an energy scavenging device. This means the amount of energy is limited, and in the case of purely battery operated devices finite. Next to limited energy the microcontroller has limited programming resources like RAM and ROM memory. This limits the complexity

Device	Mode	Power consumption
Radio	Transmit -25dB	8.5mA
Radio	Transmit -10dB	11mA
Radio	Transmit 0dB	17.4mA
Radio	Receive	18.8mA
CPU	active (4MHz, 3V)	2.240mA
CPU	LPM3 (32KHz, 3V)	1.9μA
CPU	LPM4 (0Hz, 3V)	0.5μA
Accelerometer	On (3.3V)	528μA
Serial Flash	Read (3.6V, 20MHz)	10mA
Serial Flash	Write (3.6V)	17mA

Table 1.2: *Power consumption statistics of various devices on the BSN [Lon] nodes. The device power consumption information is taken from the datasheet of the CC2420 [Ins07] and the MSP430F149 [Ins04] microcontroller.*

of the programs that can be implemented on the node. The processor on the node has limited speed itself, and usually runs in the range of 1 to 10MIPS.

1.2.1 Microcontroller resources

The microcontroller used in wireless sensor nodes has limited memory space and computational speed. Microcontrollers are fully integrated systems on a single chip, the chip itself is limited in size as chip size determines price. The limited chip space determines how much RAM and ROM are integrated.

The limited RAM and ROM space mean that the software complexity is limited. There is a high pressure on RAM usage as the MAC and network layer are very dependent on this for their receive and transmit buffers.

Most microcontrollers used in wireless sensor nodes range in speeds from 4MHz to 20MHz. This limits the computational complexity of the software. Most processors have features for dynamic frequency scaling but usually the processors run as fast as possible to reduce the active time of the processor.

1.2.2 Power

The biggest constraint on wireless sensor networks is the power consumption, without power the node will not function so this determines node and network lifetime.

Table 1.2 shows how much power the devices on the BSN nodes consume. The radio chip is the top consumer of power of all the devices on board. This means that most of the lifetime of the wireless sensor network is determined by the efficiency of the MAC protocol that controls the radio. This means the MAC needs to be efficient and reduce the amount of on time of the radio. Another method in reducing the power consumption is by reducing the transmission power of the radio, this can be done if the target neighbor is nearby.

The microcontroller consumes much less power compared to the radio but is on much more often then the radio to do processing or waiting for events. The controller power can have a large impact if the duty cycle of the radio is very small. Effective use of the various sleep modes of the microcontroller can reduce the power consumption as is evident between

the difference in sleep mode power consumption.

1.3 Thesis work

Now that we are familiar with the terms used in wireless sensor networks and that we have a bit contextual information we can explain what this thesis is about.

1.3.1 Motivation

The Technical university of Eindhoven has its own wireless sensor application framework (OSAS [BLV09]) combined with the operating system TinyOS [PLC03] running on the BSN hardware [Lon]. This platform has a some large shortcomings that prevent its practical application in various scenario's:

- Life time of the network is at most one day, this is insufficient for long term tests.
- There is little available resources left with the current software stack. This limits the application complexity.
- The traffic on the wireless sensor network can vary a lot, there are no provisions to accommodate these variations.
- Changing the underlying operating system or MAC protocol is cumbersome and takes a lot of effort to achieve.

1.3.2 Research questions

As we now know what problems we face we present the research questions.

The research questions are

- Can we reduce power usage?
- Can we reduce resource usage?
- How does power consumption change if the system is traffic aware?
- How do you abstract MAC functionality without losing functionality?

1.3.3 Thesis focus

This thesis focuses on the improvement of a few components in the software stack that runs on a wireless sensor network node. Figure 1.3 shows what layers generally exist in a wireless sensor network. Figure 1.6 shows what layers are modified in this thesis.

This goal is to improve power efficiency of the wireless sensor nodes. The main culprit of high power consumption is the lack of a good MAC layer. To improve power efficiency a new MAC (TRAWMAC [JAM08]) is implemented on the wireless sensor nodes.

To improve the flexibility of the system an API layer is inserted between the MAC and the layers above that abstracts MAC functionality. This way the higher level layers become independent of MAC implementation.

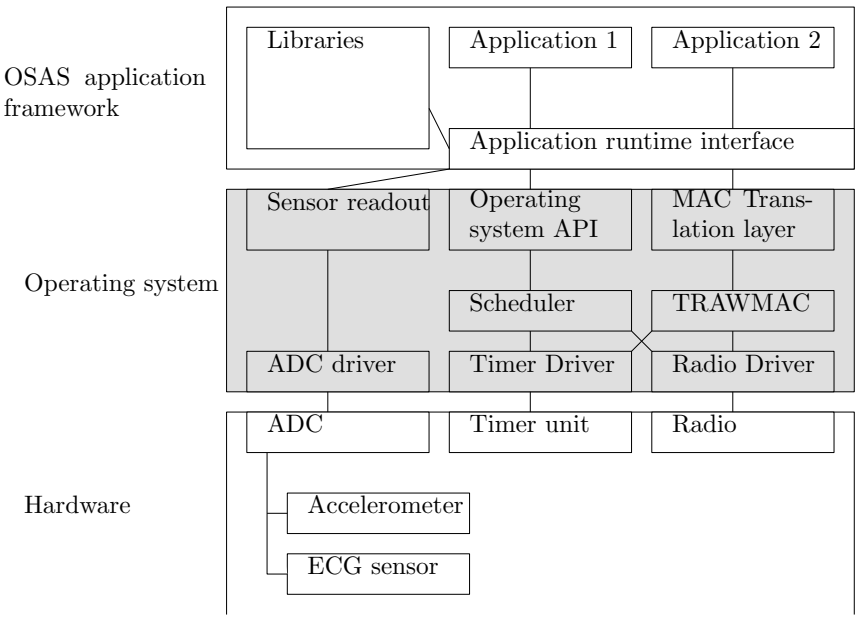


Figure 1.6: *In this figure we show the changes made in the software stack of the wireless sensor network. The gray parts are where the changes are made and where new layers are inserted.*

Next to the new MAC and API layer a new operating system was developed to reduce power consumption even more by utilizing the processor sleep modes. The current operating system tinyOS [PLC03] consumes a lot of RAM memory for various buffers. The choice was made to create a new, simple operating system supported by drivers for the peripherals like sensors and the radio. The goal of this operating system and its drivers is to be RAM memory efficient.

Chapter 2

MAC protocol

Goal of this chapter is to explain the TRAWMAC protocol that we have implemented. We will give a small introduction on what different MAC protocols exist and what research has already been done before we continue on to the TRAWMAC protocol.

2.1 Taxonomy of MAC protocols

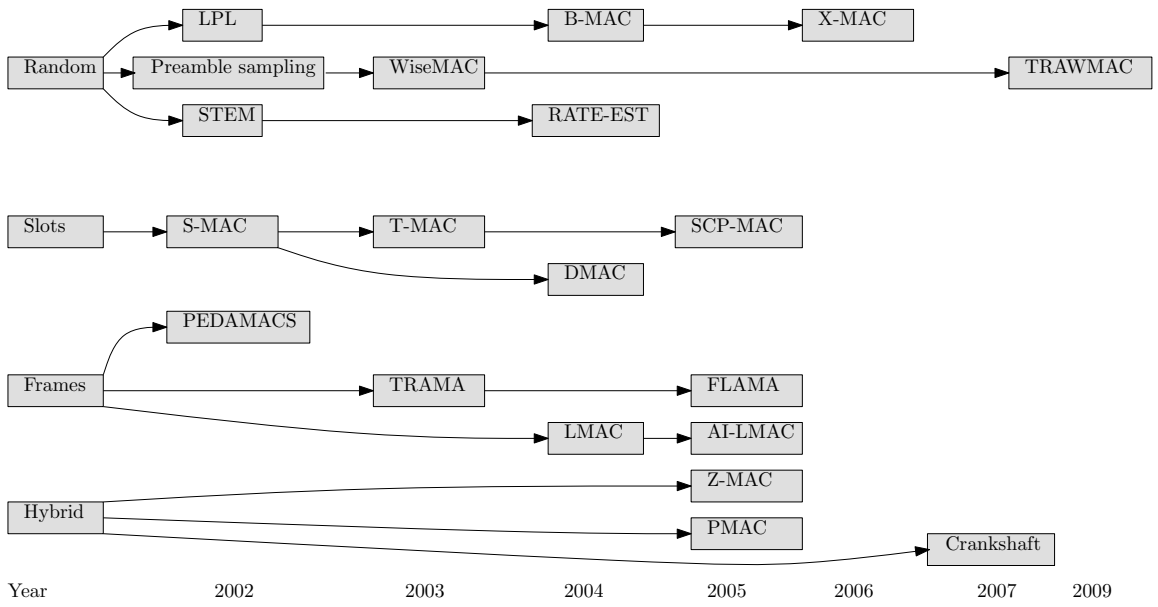


Figure 2.1: *Taxonomy of various MAC protocols adapted from the MAC chapter by Langendoen et al [Lan07]. In this list the four main classes of MAC protocols is shown with the corresponding protocols. The protocols are organized by publication date and relationship.*

MAC protocols for wireless sensor networks are an active field of research. There is large *alphabet soup* of MAC protocols as evident from survey from langedoen et al [Lan07].

The reason for the proliferation of MAC protocols is that there are many methods in saving energy, each with their own requirements in software and the sensor node hardware.

Many MAC protocols are also built to optimize usage with a specific application.

There are three main classes of protocols, we will briefly explain what the idea is behind each protocol class. There are also *hybrid* protocols that borrow some concepts of one of the three classes. This subdivision of protocols is borrowed from the paper of Langedoen et al [Lan07].

- Random access protocols
- Slotted access protocols
- Frame based access protocols
- Hybrid protocol

The random access protocols is where the nodes choose when to listen and when to sleep. The advantage of this setup is there is no explicit synchronization step required and no mutual time keeping needed, this keeps protocol complexity low. The disadvantage of such a protocol is that energy is wasted due to waking up when no transmission is occurring, and collisions between transmissions.

Most of the random access protocols differ in how they signal when data is ready. There are protocols that send out a *busy tone* to tell one specific target that data is ready, this is called *preamble sampling* Protocols based on this mechanism are [EHD04] and the extension TRAWMAC [JAM08]. Another novel approach is when using a special low power receiver to detect the busy tone on the medium. This concept is called *wake up radio* and is used in the Picoradio project [CG01].

The slotted access protocols is where the nodes agree on a common sleep and wake up pattern. The channel access is based on contention as is with the random access protocols. But now the chance for a collision becomes much higher as all traffic is grouped into a much smaller time domain. The prime example of these protocols is the S-MAC protocol [YHE02].

The frame based protocols is where the node access to the medium is partitioned into *frames* and exactly scheduling what node is allowed to transmit/receive on what slot, this principle is also known as TDMA (Time Division Multiple Access). The main advantage of this class of protocols is that no collisions can occur (as frames are not overlapping) and overhearing is greatly reduced. The disadvantage is that synchronization protocol must be implemented to partition the slots and a distributed schedule must be built. Timekeeping is very important in frame based protocols, so an accurate clock must be present to maintain synchronization between all the nodes.

There are a few protocols that counteract the problems of frame based protocols. The PEDAMACS protocol [SCV03] solves the problem of scheduling by designating one *base station* that maintains synchronization in the network. The base station can reach all the nodes in the network, but the nodes do not have to reach the base station. The TRAMA [VR03] and FLAMA [RGLAO05] protocols use a distributed solution for determining schedules of nodes. In the TRAMA protocol the schedule is created in a distributed fashion. Additional to creating a schedule, in the TRAMA protocol the traffic loads of the nodes are taken into account in the schedule. This is done by periodic advertisements of the node's traffic load. As the protocol is traffic aware nodes that transmit a lot of data will get more slots to transmit and receive data. The FLAMA protocol does away with the periodic broadcasting of traffic loads and uses an explicit mechanism to advertise the traffic slots to one neighbor.

The LMAC protocol uses a different mechanism in determining slot scheduling by sending each transmission slot data about its internal schedule and an bitmap that shows what slots are occupied by its one hop neighbors. The nodes can compile a map of occupied slots and free slots of their neighborhood.

The hybrid protocols mix and match a few features of two or more protocol classes.

A few MAC protocols with good power efficiency only handle a small range of traffic loads and applications. This can be a problem when a protocol needs to handle a wide range of traffic loads and characteristics. There are some protocols that can adjust internals to adapt to various traffic loads. Most of them are protocols taken from one of the three main MAC protocol classes. These protocols are called *traffic aware*.

The TEEM protocol [CS03] is based on S-MAC protocol and is a member of the slotted access class of protocols. The S-MAC protocol fixes the active slot of all the nodes but in the TEEM protocol, the active slot can change. The time between active slots can be increased if the traffic load is small or the tolerable latency is high. The time between active slots can be reduced if the latency needs to be low for the traffic. In both cases the tradeoff is made between traffic and power consumption. When you allow more traffic you will spend more energy.

The TRAMA protocol [VR03] is a member of the frame based protocols. In frame based protocols the medium is divided up into time slots with each has an owner. In TRAMA nodes that need more slots as they have more traffic can allocate more slots to themselves. This way the node can satisfy its traffic requirements.

The TRAWMAC protocol [**?**] that is presented here in this thesis, is a member of the random access protocols. It is based on the WISEMAC protocol and extends it with traffic aware functionality. In random access protocols the nodes wake up and sleep at regular intervals. The distribution of when those nodes wake up and sleep is random. In the TRAWMAC protocol, the nodes can reduce their sleep time to allow more traffic coming in. This change of the sleep time can happen dynamically.

In the following chapters we will explain our implementation of the TRAWMAC protocol in more detail.

2.2 Introduction

The TRAWMAC(TRaffic AWare Media Access Control) protocol was developed by Ansari et al [JAM08] at RWTH Aachen University. The goal of this protocol was to provide energy efficient operation on a wide range of operational conditions.

The protocol uses the preamble sampling method (see section 1.1.3) for achieving power efficient operation. The protocol itself is able to adjust to various traffic loads and provide power efficient operation under these changing conditions.

2.2.1 Protocol features

As the name TRAWMAC implies it is traffic aware. The protocol can adjust operational parameters dependent on the traffic pattern it experiences, these adjustments are controlled by the application. This means the application needs to be aware of operational parameters of the traffic, like latency and throughput. The result of this traffic awareness is better power efficiency under varying traffic loads.

The protocol by design is also aware of its direct neighbors, these are kept in a list and are accessible by the higher layers (application and or router). The router can use this neighbor information to discover the network topology and build routes to other nodes. Neighbor information is also usable when determining network density, transmission power and other parameters that depend on local node density.

2.2.2 Design choice

The main reason TRAWMAC is chosen for its ability to efficiently deal with wide a range of traffic conditions. The reason for these varying traffic conditions is that the application is actually an application framework that generates a wide range of traffic patterns. The ability of changing MAC properties to accommodate differing traffic patterns is not an unique feature as the TEEM [CS03] provides more or less the same functionality but is built on S-MAC. Better power efficiency can be achieved with preamble sampling protocols like WISEMAC [EHD04] then S-MAC as seen in the paper of WISEMAC.

The application framework OSAS [BLV09] runs applications that are dynamically loaded during the lifetime of the wireless sensor network. The wide range of applications demands traffic flexibility of the MAC protocol to ensure network lifetime. The current system has insufficient network lifetime to be of practical use. The use of this protocol ensures much longer lifetime of the network, exact results are given in chapter 6. The application framework has information about the traffic patterns it generates, this can be used to setup TRAWMAC.

2.3 TRAWMAC

In this section we will explain how the TRAWMAC protocol works, how the power efficiency is retained during all traffic patterns and how the neighborhood exploration is performed. For each part of the protocol we will explain some implementation details.

2.3.1 Preamble sampling

Most MAC protocols use periodic operation of receiver and transmitter to achieve power efficiency. TRAWMAC uses a scheme that is derived from the WISEMAC [EHD04] protocol, this scheme is called *Preamble Sampling*.

In preamble sampling all the nodes periodically perform a CCA (Clear Channel Assessment). The interval in which the nodes perform a CCA is called a CI (Check Interval). The CCA is performed by turning the receiver on, check if some transmission is active, and if the data is not for the node, going back to sleep.

When a transmitter wants to transmit data to a node, it will transmit a short preamble (series of small frames) before the actual data. The duration where MFP's are transmitted is as long as the CI of the receiver, this way you are sure that the receiver has received the preamble and is ready to receive the data.

This mechanism is duplicated in TRAWMAC with some changes. TRAWMAC calls the preamble frames MFP (Micro Frame Preamble) and DFP (Data Frame Preamble). The MFP frame is the short preamble transmitted to a target neighbor, the neighbor samples the medium and receives this preamble. At the end of the CI of the target neighbor the transmitter transmits the data destined for the target. See figure 2.2 for a graphical representation of this mechanism.

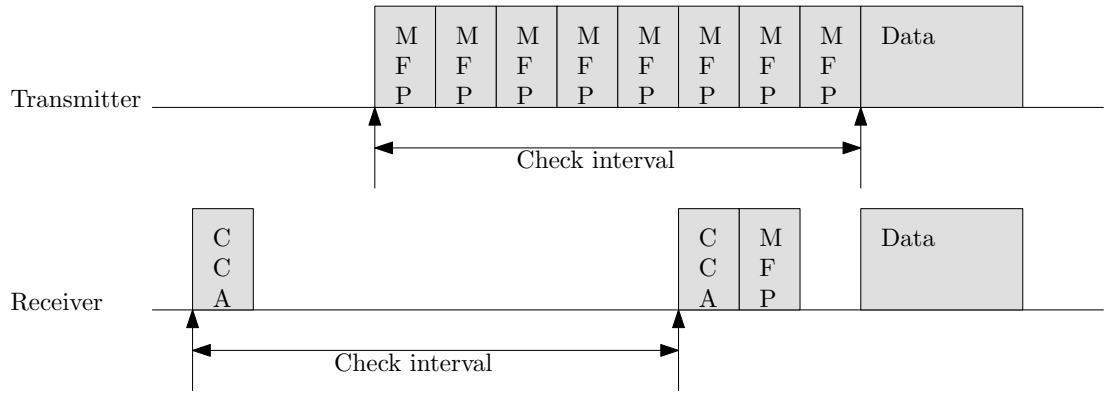


Figure 2.2: *Micro Frame Preamble scheme of TRAWMAC*

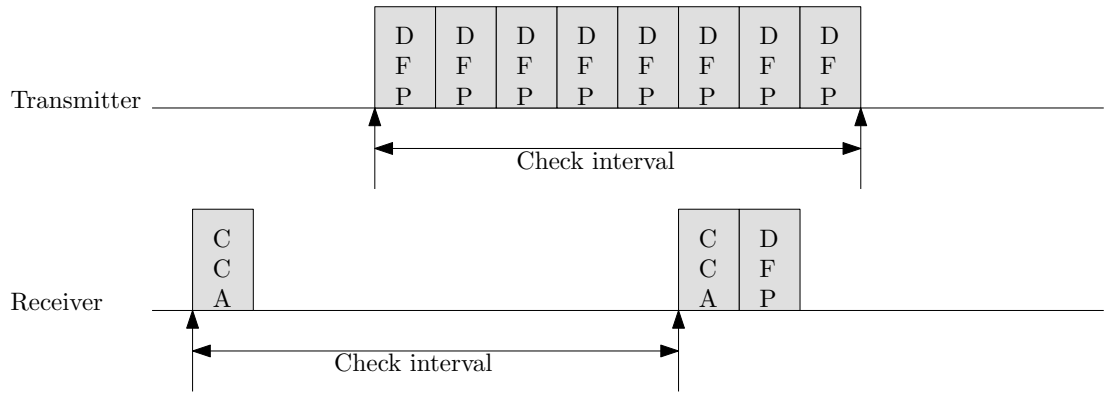


Figure 2.3: *Data Frame Preamble scheme of TRAWMAC*

When the data to be sent to the target neighbor is small the MFP mechanism introduces a large overhead for a small amount of data. The DFP counteracts this problem by adding the data to the MFP, this combination of MFP and data is called a DFP (Data Frame Preamble). The DFP scheme reduces the overhead of small data transmissions.

The problem with preamble sampling schemes is that a lot of transmissions are done that are not received by any neighbors. This is a significant waste of energy. One method in counteracting this waste is that the receiver acknowledges the MFP or DFP packet. This way the transmitter knows that the receiver is on and ready to receive the data. This improves the energy efficiency but does increase complexity and reduces robustness of the protocol. The reduced robustness is when at the end of the whole cycle the data is transmitted and is corrupted there is no retransmit mechanism.

Traffic that is directed to one address is also called unicast traffic and traffic directed to all neighbors is called multicast traffic. The DFP and MFP communications can be used for both types of traffic but with multicast traffic it is not possible to use acknowledgements. Using

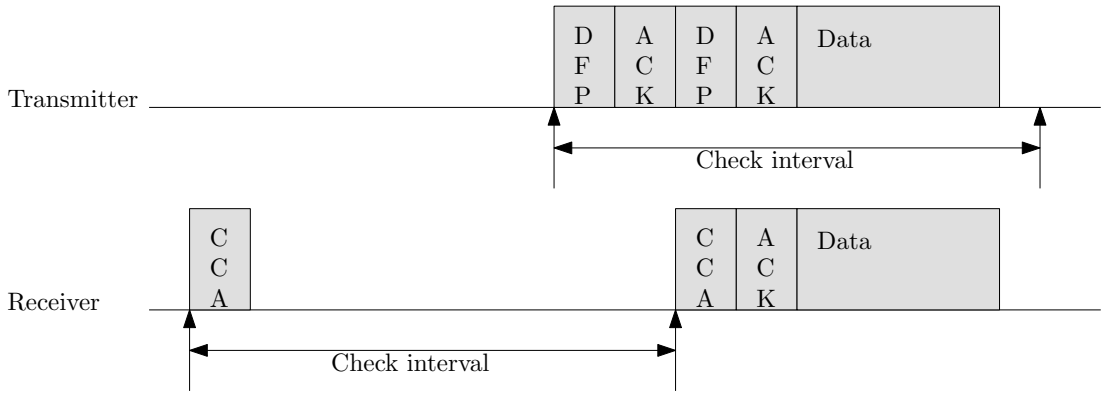


Figure 2.4: *Micro Frame Preamble with acknowledgements*

acknowledgements on multicast traffic would create produce lot of colliding ACK packets and individual handling per neighbor, consuming lot of power and memory.

The TRAWMAC protocol adds one extension to the preamble sampling principle. The CI can be modified dependent on the amount of traffic. This means that the CI can be reduced when more traffic is anticipated or increased when very little traffic is expected.

2.3.2 Implementation

TRAWMAC is implemented on the BSN motes [Lon] developed by the Imperial College London. The hardware used by the protocol is the CC2420 transceiver and one timer from the MSP430 controller on de node, the timer runs at 32768Hz.

The radio is controlled by a small driver that handles initialization, transmission, reception and the various radio sleep modes. Data ready for transmission and received data are passed through allocated buffers from the MAC protocol. The passing of allocated buffers between the application layers is also called zero copy semantics. The radio has multiple operational modes for transmission, reception and various low power modes.

The timer functionality is abstracted by the scheduler of the operating system. The scheduler executes periodic tasks stored in a task list, the scheduler uses a FPNS (Fixed Priority Non preemptive scheduler) scheme. The TRAWMAC software runs as a task that is executed by the FPNS scheduler 1024 times per second. The operating system also offers general purpose counters that count down and signal when they have elapsed. This is used by the protocol software for timeouts and periodic activations. The TRAWMAC protocol is implemented as a state machine that runs the protocol logic. The state machine is comprised out of two parts, one for the reception of data and one for the transmission. The reception part is entered when the node is scheduled to listen for preamble packets. When a MFP is received, a sequence of actions is performed to receive the data. The transmission part of the state machine is activated when there is no listening active and there is data pending in the transmission queue. First a delay is performed until the time the transmitter thinks the receiver is online. The Transmitter then transmits an MFP, switches the radio to receive mode and waits for an acknowledge. If this does not occur another MFP is transmitted. This

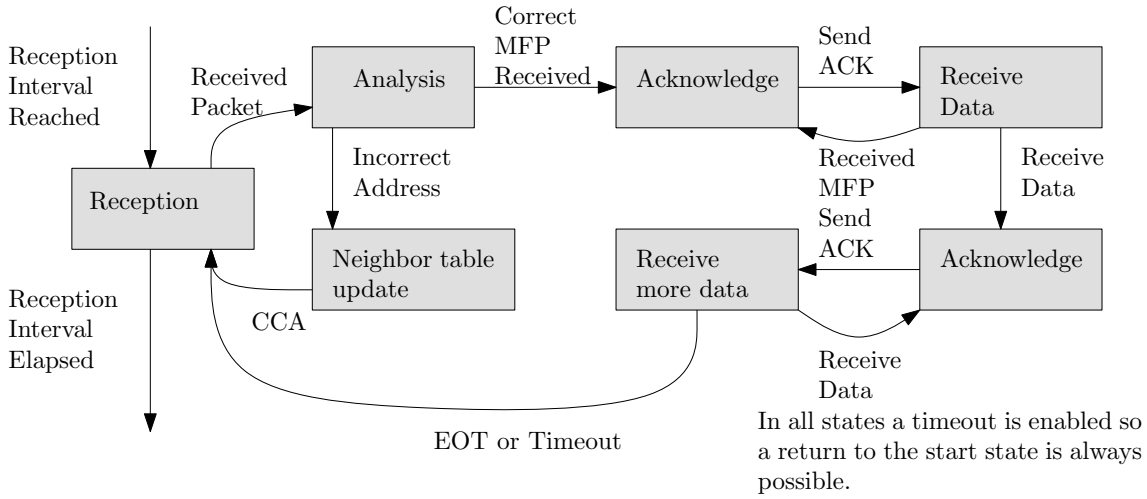


Figure 2.5: *This state machine shows how TRAWMAC handles reception of data addressed to this node. The reception sequence started at a detection of a MFP addressed to the current node. At any place in the reception states a timeout can occur so the stack returns into a valid state for the next CI. The timeout duration is set to a small constant value.*

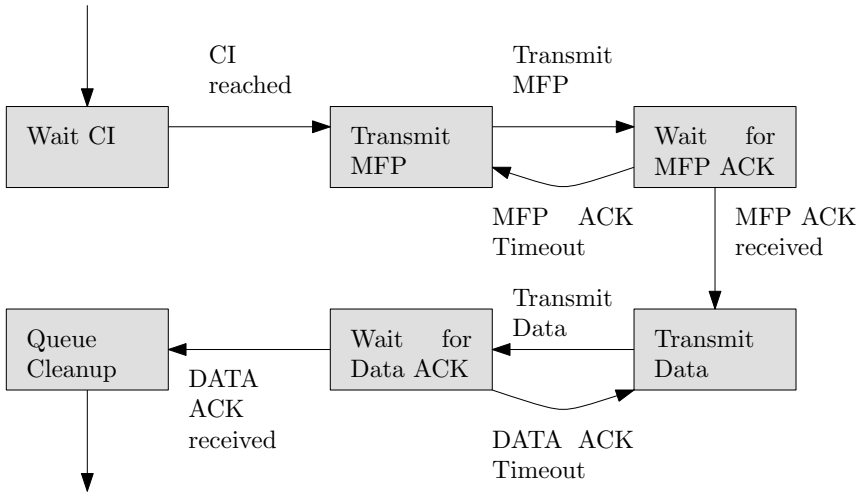


Figure 2.6: *This statemachine shows how TRAWMAC handles transmission of data. The transmission sequence is started when a packet is pending in the transmit queue. At any place in the receive section a timeout can occur so the stack can return to a state where it can restart operation.*

process repeats until an acknowledge is received or the whole process takes longer then the CI of the receiver.

As all clocks, the clocks on the BSN motes drift a bit. This drift is not a big problem as all preamble sampling based protocols can tolerate this. The reason for that is preambles are sent during the entire CI of the receiver. Even if the receiver has drifted by a considerable

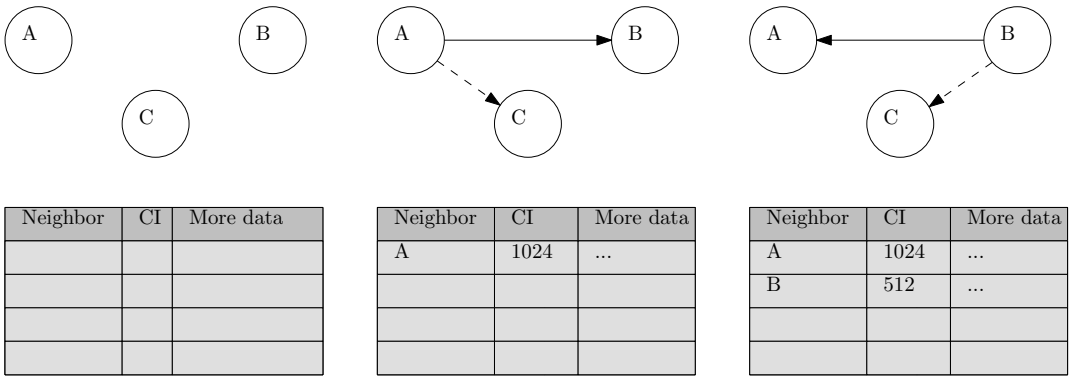
Type	Name	Description
uint16_t	Type	Frame type
uint16_t	src_addr	Source address
uint16_t	dest_addr	Destination address (Not present for Acknowledgements)
uint16_t	src_rx_period	activation interval of the source
uint16_t	src_rx_act	pending time for activation

Table 2.1: *MFP, Acknowledgements and Beacon structure*

amount of time, it will eventually receive the preamble. The rule is, the less the clocks drift apart, the less power is wasted by sending preambles.

2.3.3 Neighborhood detection

The transmitter needs to know what the CI's are of the other nodes. The CI tells when the destination is listening to information as it is only periodically on. There are a few mechanisms the nodes can use to get this information.



(a) New node added **(b)** Listening to node A, add entry. **(c)** Listen to node B, add entry

Figure 2.7: *New nodes and other nodes that receive surrounding chatter from other nodes use the information that is in each MFP and ACK to populate and renew their neighborhood table.*

Each node has a period it listens to the medium to receive preambles. The node can receive packets that are not destined for the current node. The MFP and acknowledgement packets store data about the CI and the time until the next CCA (see table 2.1) . Nodes that overhear this information can store this in a table of neighbor CI durations. When the data already exists for what specific node it is updated.

Next to overhearing traffic on the network the nodes also emit periodic beacons, these beacons contain scheduling information of the transmitter. This is useful for networks that do not see a lot of traffic. The beacons provide a source of traffic that can be used to synchronize with the neighbors.

In the case of very little communication, the nodes periodically transmit a beacon that has their CI and time until next CCA, nodes overhearing this data can add this to their neighbor table.

When nodes first start up, they will enter a so called *probe mode* where they will continually have their receiver operational to detect any transmissions. The packets with neighborhood information is recorded in the neighbor table and when sufficient time has elapsed or enough neighbors have been collected. The node enters normal operation and advertises its existence via beacons.

2.3.4 Traffic awareness

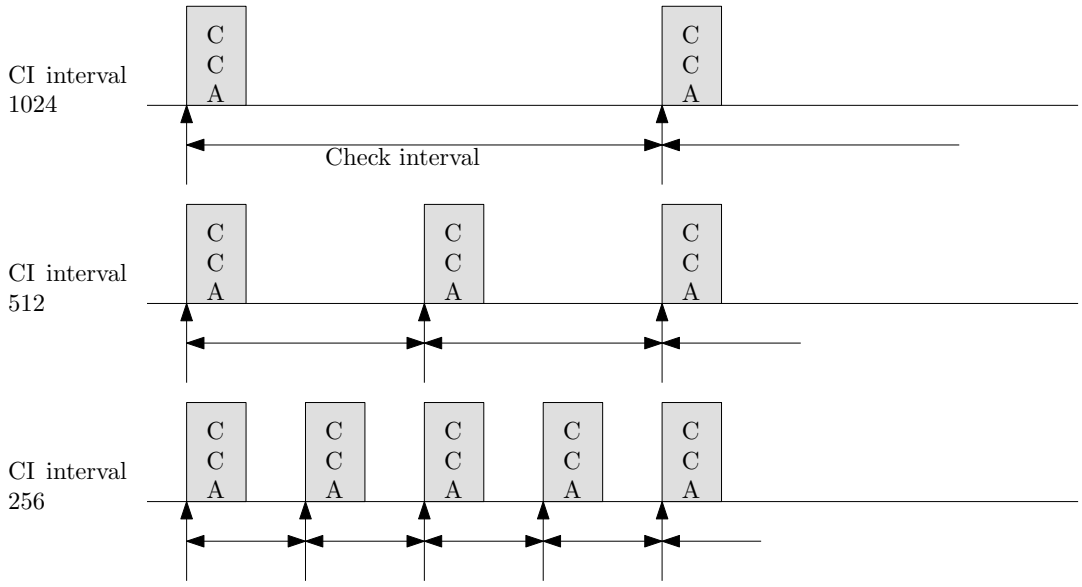


Figure 2.8: *Check Interval (CI) adjustment in TRAWMAC happens always at a power of two times the current CI. This creates extra time slots while keeping in synchronization with the other nodes that have not updated to the new CI.*

As the TRAWMAC protocol has traffic awareness functionality, this must be exposed to the application or traffic analysis is required. In our application, the OSAS [BLV09] application framework has timing information embedded in the services that OSAS delivers. This timing information can be used to find out at what frequency data is going to be transmitted. This information can in turn be used to determine the length of the CI of the TRAWMAC protocol. We have created a more general solution to forward timing information to the MAC by using a MAC API. This API is described in chapter 3.

In our implementation we have added some constraints in the CI manipulation. The CI will be changed to the nearest multiple of the current CI. For example: if the current CI is 1024, and a CI of 200 is requested, the CI is changed to the nearest power of two times the current CI, in this case this is 128. For graphical example, see figure 2.8

The reason for this is that when the CI gets reduced, neighboring nodes have not updated their values. The receiver wakes more often now but still remains awake in the same periods the neighbors expect. When the neighbors communicated with the node, they will see in the communication packets that the CI of the node has changed and update their neighborhood table accordingly. When increasing the CI, the neighbors will expect a wake up period where

there is none. This results in timeouts, when communication is reestablished the neighbors update their neighbor tables with the new interval.

2.4 Problems faced

During the implementation of TRAWMAC I faced a few problems.

There was little hardware documentation available of the wireless sensor nodes. This proved to be a big problem during the development of the protocol and the underlying operating system as it was sometimes not clear if bugs where due to hardware issues or programming mistakes. After we got the schematics of the nodes we could fix some severe issues with the protocol. Communications via the bus to the radio chip had to be very slow as at higher speeds errors occurred. After we found the reason for this (improper initialization of devices on the bus) we could increase the speed of the communications to and from the radio, resolving a lot of timing issues with TRAWMAC.

Some aspects of the TRAWMAC protocol where insufficiently described. It was unclear how neighborhood propagation was done, so our own solution was implemented and proved to be satisfactory. See section 2.3.3 for a more detailed description.

In the documentation of TRAWMAC [JAM08] it was mentioned that the CI's could be changed to accommodate for changing traffic levels. How this change was performed was unclear and that it was bound to a few values was also not described. We have created our own constraints where the CI is bound to powers of two times the original CI, see section 2.3.4.

The CC2420 radio chip is made for the 802.15.4 MAC protocol and provides specific features for this. This chip was not a good fit for TRAWMAC as there was no information on durations of changing to different modes. This was fixed by allowing a part of slack around the communications, this does mean there is some energy waste in communications. We propose a solution for this in section 6.10.

2.5 Missing functions

The protocol as described here has not been fully implemented. This is due to lack of time and lack of a suitable set of test cases.

The following list shows what parts have been not been implemented:

- Data Frame Preamble (DFP) communication is not implemented.
- Incomplete neighbor table handling, no decay or replacement rules.

The data frame preamble from figure 2.3 is currently not implemented. Implementation of the DFP communications is not too complicated but would increase complexity of the TRAWMAC state-machines for reception and transmission. The code for the MFP communications can be copied verbatim and the last phase of data transmission omitted. Some additional rules in the transmission start needs to be added to decide when to do a MFP or a DFP transmit. This depends on the size of the data to be transmitted.

The MAC handling of the neighbor table is incomplete. Neighbors are added but never removed from the table. Some decay or replacement rules need to be added. Using an Least Recently Used scheme in replacing old neighbors is one method of removing neighbors. This would require some changes in the update and add routines of the neighborhood table

handling. These functions are used in many parts of the MAC so it is important that these terminate quickly.

Chapter 3

MAC Programmer Interface

The goal of the MAC API is to create an common suitable interface for many existing MAC protocols while still using additional MAC functionality.

3.1 Introduction

The MAC API is an application programmer interface for controlling MAC layers in wireless sensor networks. The API offers a range of functions for controlling the functionality of MAC protocols and abstracting implementation or network details away from the application. This creates a common interface for a wide range of MAC protocols.

3.2 Related work

There has been some research in creating an API for communications between the application layer and the MAC protocol. Some software stacks completely abstract away the details of a wireless sensor network by creating an application framework. Examples of these are TinyDB [MFHH05], MATE [LC02] and OSAS [BLV09] developed here at the university. The goal of such an application framework abstraction is that it makes it easier to program wireless sensor networks without dealing with networking, program distribution and communication.

One other instance of an layered API is the B-MAC protocol [JP04]. This protocol is actually an API above some MAC level mechanisms like preample sampling and buffering. It only provides this functionality and leaves it up to the programmer to create higher level functions. It basically separates the MAC into two parts, one timing critical part that deals with the radio and a more abstract part that deals with internal structures that is not timing critical. The timing critical part of the MAC deals with the radio architecture while the non timing critical can use an uniform interface. The goal of this layering is to abstract radio control details while making it easy to implement different MAC protocols.

3.2.1 Advantages

The API has a few advantages for the whole platform.

The wireless sensor network implementer can change MAC protocols with little effort. Not all MAC protocols are created equally and each has its advantages and disadvantages. By making the change in MAC protocol easy, the implementer can optimize the performance

Latency	Throughput	Periodic	Application
Low	Low	No	Alerts and alarms
Low	Low	Yes	Low frequency real-time monitoring
Low	High	No	Sampling on alert
Low	High	Yes	High frequency real-time monitoring
High	Low	No	Low frequency requests
High	Low	Yes	Low frequency sampling
High	High	Yes/No	Data logging / burst transmission

Table 3.1: *Traffic patterns in wireless sensor networks classified by latency tolerance, throughput and periodic traffic.*

of the wireless sensor network. The easy change of MAC protocols is also a useful feature in research applications. Developers of new MAC protocols can more experiment more freely with various protocols and observe the improvements. As only the MAC layer is changed the rest of the application can stay more or less the same, influencing the measurement as little as possible.

Another advantage is by providing a common interface that provides extra functionality besides simple data transfer functions like receive and transmit. Many MAC protocols provide additional features or parameters to improve efficiency, but if the application cannot manipulate these, these advantages are lost. The application can also receive additional data from the MAC layer to perform additional functionality like power awareness, routing and other functions that depend on network wide information.

3.3 Traffic patterns

Before we start going in depth on the MAC API, we need to find out what kind of traffic the MAC API should support. Various applications create various patterns of traffic. The MAC API makes it possible to accommodate these traffic patterns by explicitly supporting this.

3.3.1 Traffic patterns and applications

There are many applications for wireless sensor networks, but the types of traffic they generate fall into a few categories. We have looked into some practical use cases from the WASP project [con09] and found some common patterns of communications. These application classes are summarized in the table 3.1.

The difference between these traffic classes is the amount of traffic (Throughput), Latency tolerance (how fast the traffic needs to arrive) and if the traffic is periodic (produced at regular intervals). Most traffic patterns have requirements that can be fulfilled by the MAC layer, and to a lesser amount the network layer. The MAC layer determines the maximum throughput, latency and periodic behavior. These properties influence how well they execute a traffic pattern in table 3.1.

The traffic patterns emerge from the application the wireless sensor network is used. In high speed real time monitoring applications the latency must be low, you want minimal delay between measurement and display. These applications are usually monitoring things that change in time quickly or where a high reaction time is important. These are applications

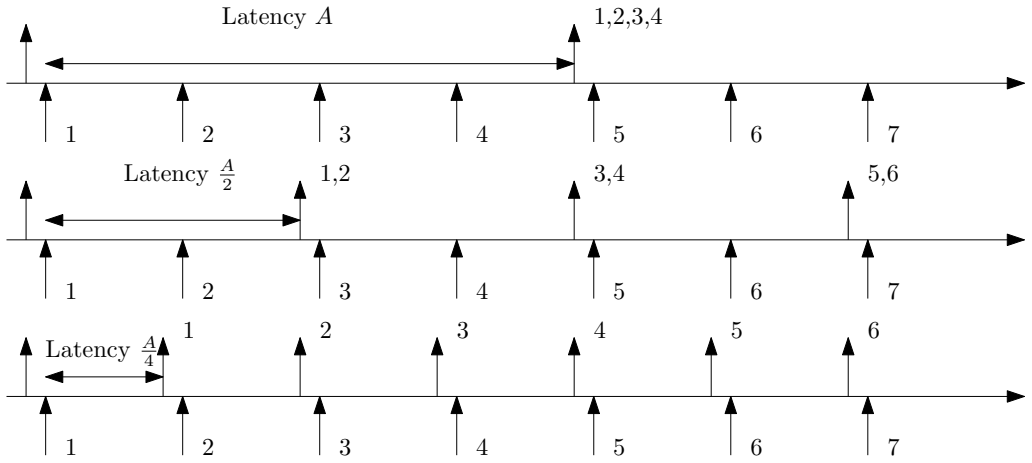


Figure 3.1: *Influence of duty cycle on the packet latency. The shorter the duty cycle the shorter the time that the packet is offered for transmission and and actually transmitted.*

like medical monitoring of heartbeat or industrial process control. High throughput usually comes from sensors that produce a large amount of data. This is the case for image data pictures and movies, fast sampling for ECG's or sound.

3.3.2 Pattern requirements

Each traffic pattern has some requirements in throughput, latency tolerance and periodic behavior. To fulfill these requirements a MAC protocol needs to be selected that fits the application requirements. These are communication requirements like the bandwidth, latency and periodicity but operational requirements like energy efficiency. There is a tradeoff to be made between power efficiency versus throughput and latency. MAC protocols in generally use periodic sleeping of the radio to improve power efficiency, the longer the sleep period, the better the power efficiency. The disadvantage is that latency is increased and throughput is reduced.

Many platforms support on the fly changes of functionality of the wireless sensor networks, so choosing one MAC protocol may not be the best solution for all possible applications. Traffic aware protocols that optimize power efficiency while keeping the latency/bandwidth requirements are a good match to these applications. But the protocol needs information about changes in operational requirements of the network.

The TRAWMAC protocol can change its parameters to be efficient with different bandwidth and latency settings. Many other protocols can be adapted to change operational parameters to provide better energy efficiency under various traffic patterns. The OSAS application framework sets the operational parameters of the MAC protocol according to the requirements of the installed applications/services on the OSAS runtime. When new applications are installed on a node via OSAS they are usually accompanied with temporal or bandwidth information, this can be used to setup the MAC protocol.

As almost all MAC protocols use a scheme of sleeping the radio when not in operation,

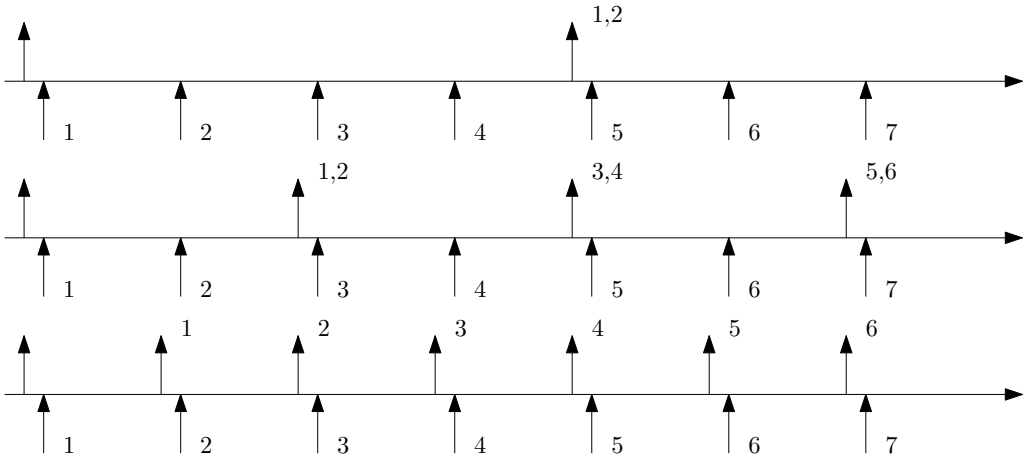


Figure 3.2: *If we assume that each transmission slot allows only two packets to be transmitted, then the amount of slots limit the amount of packets that can be received or transmitted.*

this is done to preserve power. The length of this duty cycle determines latency and in many protocols also available bandwidth. In figure 3.1 is shown that latency increases when duty cycle increases, it takes longer for the data to be transmitted thus a longer waiting time is required. There are some MAC protocols that only allow one or a limited amount of packets to be transmitted each period the radio is awake, this sets a hard limit to the node bandwidth. Other MAC protocols like S-MAC [YHE02] allow communication between two parties beyond the active period to transfer additional data. For an graphical example see figure 3.2.

3.3.3 Network state information

Next to support for traffic patterns for the application, the MAC API can provide information from the MAC. The network layer has usually a router component to route data over multiple hops. The MAC layer usually keeps information about neighbors like timing information, reception strength, timeouts and more. The router in the network layer can use this information to make better informed routing decisions.

3.4 API

Now we know what traffic patterns exist and what properties define these patterns. A MAC protocol that just provides support for transmission and reception can not support all patterns while preserving power efficiency. The MAC requires additional information about the transmission and the application state so that the MAC can anticipate the traffic pattern. Not all MAC protocols support this functionality but TRAWMAC is a tunable MAC protocol and has provisions to support a wide range of traffic patterns. The API also provides information about current node neighbors, either provided by the underlying MAC protocol, or emulates this functionality.

Type	Name	Description
uint16_t	address	Target or source address
int8_t	strength	Reception/transmission strength in decibel
uint16_t	deadline	How long can this packet exist
uint32_t	reception time	Local time of reception
uint8_t	hop count	Number of hops this packet has seen
uint8_t	priority	Packet priority (0 is lowest)
uint16_t	data size	Size of data structure
uint8_t	retransmits	Number of retransmits
uint8_t *	data	Pointer to data structure

Table 3.2: *Packet structure*

Type	Name	Description
uint16_t	Duty cycle	Reception duty cycle
uint16_t	Minimal delay	The delay of communication
uint16_t	energy level	Energy of the node
int8_t	reception strength	Reception strength in decibel
int8_t	channel	Reception channel of the node

Table 3.3: *Settings structure*

3.4.1 Function classes

The MAC API is subdivided in multiple function classes, these are: communications, settings and neighborhood. The communications class provides functions for transmitting and receiving data to a specific address or a broadcast address. In addition to just the data there are extra fields to provide timing information, deadlines, amount of deadlines. The settings class provides methods for changing the MAC settings on a global level instead of a per packet basis, it is also used to provide data to the MAC about the operational state of the node. The information passed to the MAC consists of duty cycle, reception strength and transmission power. The neighborhood class of functions make it possible to retrieve neighborhood information of the MAC, per neighbor some information is stored about its state.

3.4.2 Data structures

The API utilizes some data structures for use in the API functions.

Communication is performed through the *packet* datatype (table 3.2). It contains all relevant information for transmission and reception of data. Some of these fields are filled in by the MAC on reception or transmission. The address field can have the address of 65535 (maximum value of a 16 bit integer) for the broadcast address. The *MAC_settings* structure (table 3.3) contains the settings that can be set in the MAC protocol. The duty cycle is the reception frequency the application expects in milliseconds. The energy level is a value that can be used by some MAC protocols to spread out power consumption through the network. The channel on which the MAC is receiving data can be set. The *neighbor* structure (table 3.4) contains all the information that the MAC has of one neighbor. It contains the neighbor address, its average reception strength, the last local time it has transmitted, its

Type	Name	Description
uint16_t	Address	Neighbor address
int8_t	strength	Running average reception strength
uint32_t	Last transmission time	In local time
uint16_t	Energy level	Node energy level
uint16_t	Duty cycle	Node duty cycle
uint16_t	Minimal delay	Node minimal delay
uint16_t	wake interval	Awake period length
uint8_t	Channel	Neighbor channel
uint16_t	TX_count	Transmit count
uint16_t	TX_fail	Transmit failures

Table 3.4: *Neighbor structure*

Type	Name
MAC_API_OK	function executed successfully
MAC_API_NOMEM	insufficient memory
MAC_API_BAD_ADDR	Bad address for transmission
MAC_API_INVALID_VALUE	Bad field value

Table 3.5: *Possible Error codes from the API functions.*

energy level, duty cycle, minimal delay, radio channel and the number of transmissions and transmission failures.

3.4.3 Calls

The function calls are used by the application to communicate with the MAC. The MAC API consists of three groups of calls: communications, settings and neighborhood.

uint8_t tx_data(packet * *data*)
This is the transmit data function of the MAC API. The data argument is a pointer to a packet structure that is inserted into the queue by the API. The function observes the priority and deadlines and inserts it in the proper position of the queue. The function returns values described in table 3.5

uint8_t rx_data(packet ** *data*)
This is the receive data function of the MAC API. The data argument is a pointer that points to a pointer to the received data structure. If the pointer remains NULL after the call, then there is no received data pending in the queue. This function returns only the MAC_API_OK value.

The communication functions are used in communication to and from other nodes in the network. The functions usually manipulate a queue of packets used by the MAC, thereby taking into account the scheduling policies.

uint8_t MAC_Setup(MAC_settings * *settings*)
This is the MAC_Setup function that sets the settings of the MAC protocol. The values

from the MAC_settings function are translated to the specific MAC and then set. There are only two responses possible: MAC_API_OK or when a invalid value was specified MAC_API_INVALID_VALUE.

void MAC_Read(MAC_settings * *settings*)
This is the MAC_Read function that reads the current settings of the MAC and places this in the *settings* argument.

The setup function applies the settings to the MAC set in the *MAC_settings* structure. The API translates the values to sensible values for the MAC as it may not support directly the value requested by the application. This translation is performed for all the possible settings in the structure.

uint8_t MAC_Neighbors(Neighbor ** *neighbors*,

uint16_t **size*)
This is the MAC_Neighbors function that returns a pointer to the neighbor pointer array and the size of this array. The values returned are MAC_API_OK or when a invalid pointer was specified MAC_API_INVALID_VALUE.

The neighbors function returns a pointer to an array of pointers to neighbor structures. These neighbor structures contain information about each neighbor the MAC is aware of, as described in table 3.4. The neighborhood structure is read only and not meant to be modified by applications.

3.4.4 Functionality

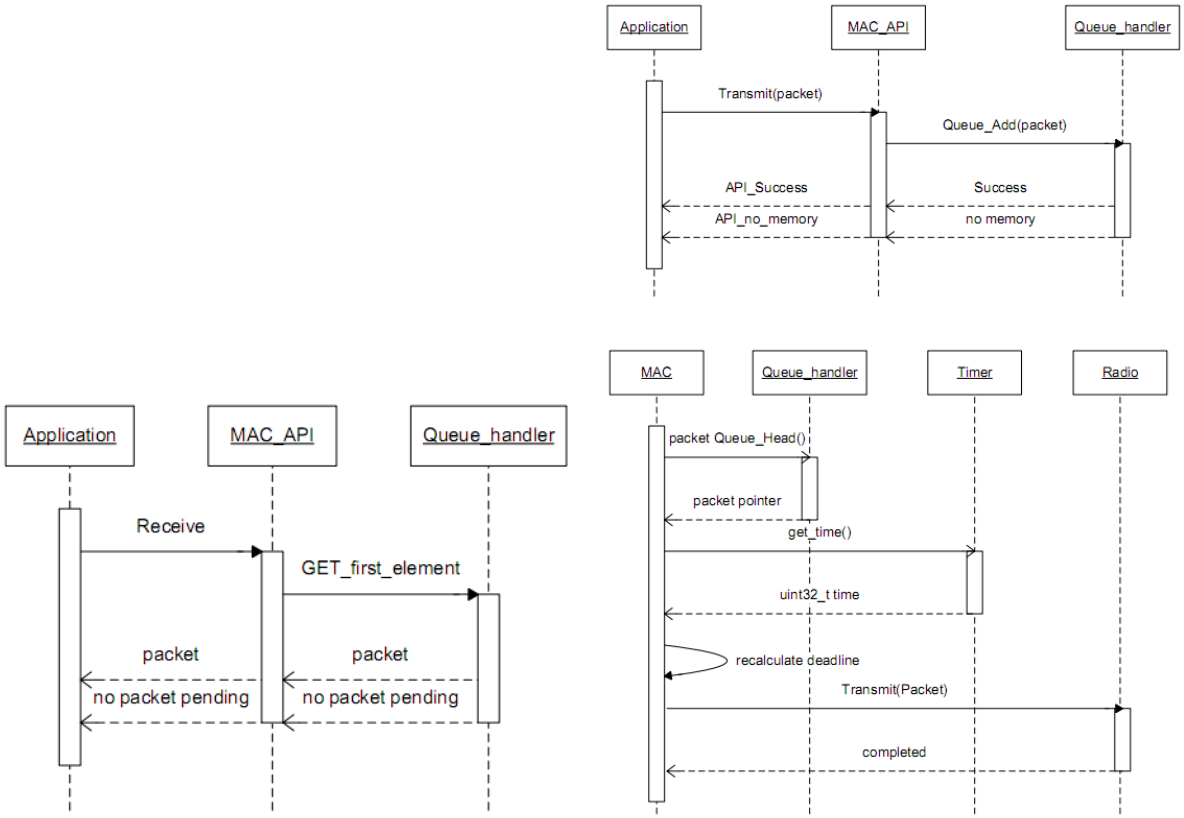
Much of the API functions are straightforward in functionality but need to be tailored to each MAC. We describe here how the MAC API interacts with the TRAWMAC protocol as no other MAC protocols have been adapted to the API yet.

3.5 Transmit and Receive

The transmit function adds data to be transmitted to the MAC transmission queue. The data to be transmitted is encapsulated in the packet data type described in table 3.2. The MAC takes the first packet in this queue structure and transmits the data to the address described in the packet. If possible it will use the fields from the packet structure to perform additional adjustments. The only field used is the deadline field where the deadline is recomputed by subtracting the time the packet has spent in the system from the deadline field.

In the reception function the data is retrieved from the incoming data queue and processed by the application. The packet data structure is the same as that from the transmission queue, so if a packet needs forwarding it can be modified and passed along in the transmit queue. The application retrieves the packet from the reception queue, and if needed frees the space used by the packet.

The currently supported fields by TRAWMAC are the address, data and data size fields, this is the bare minimum that a MAC has to support.



(a) Reception **(b)** Transmission

Figure 3.3: *These sequence diagrams show how transmission and reception performed by the application and how the MAC protocol handles this. The transmission diagram also shows how the MAC retrieves a packet from the transmission queue but that the MAC API computes the new deadline value for this packet just before transmission.*

3.6 Settings

The settings call sets the tunable parameters in the MAC that are supported. In the case of TRAWMAC the the Duty cycle parameter and minimal delay can be used to determine the new CI of TRAWMAC. This translation step is done by the MAC API to keep the constraints that are imposed on the CI (see the discussion in section 2.3.4).

3.7 Neighborhood

The neighborhood part of the API returns a pointer to a list of neighbors. This list is populated by pointers to neighbor structures or NULL pointers. The whole array needs to be searched as neighbors are added and removed by the MAC continually and some neighbors may be removed during the operation of the network. The structure is meant only to be read out by the application, writing is possible but may cause the MAC to be left in an inconsistent state or even crash.

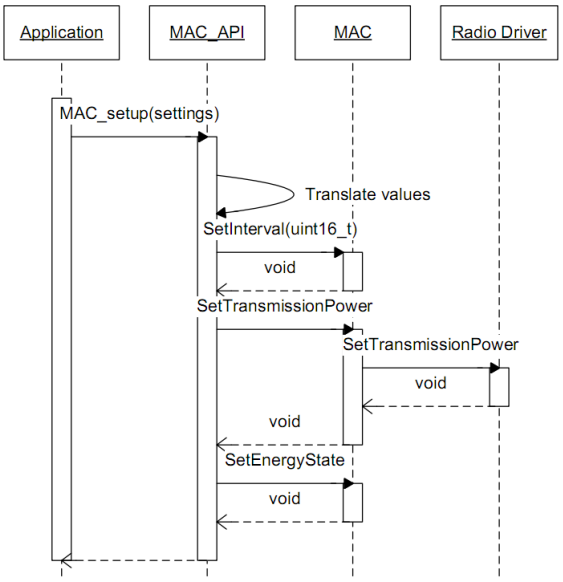


Figure 3.4: *This sequence diagram shows how the MAC settings setup is performed. The values from the call are translated to MAC specific values and then loaded through the interface functions that the MAC provides, this can differ on a MAC by MAC basis.*

Chapter 4

OsasOS operating system

The goal of this chapter is to explain how the OsasOS operating system operates.

4.1 Related work

There are a few existing operating systems in use on wireless sensor networks. The most well known are TinyOS [PLC03], MANTIS [SBH05] and Contiki [DGV04b]. Most of these operating systems are designed to be very compact and suitable for use in embedded systems. Most of these operating systems work with *Event Based Programming*, this is the mechanism where events trigger executions of tasks. MantisOS differs on this point as it also provides preemptive multitasking, but this comes at a cost in memory usage. The memory usage is optimized by using static buffers to communicate between different tasks.

4.2 Introduction

The current platform uses the TinyOS operating system to provide task functionality and hardware control. This operating system has some shortcomings in functionality and memory usage. To fix these shortcomings we have written a new operating system called OsasOS that reduces the memory usage of this operating system.

The OsasOS operating system is programmed in ANSI C and contains a small task scheduler running tasks from a fixed task list. The hardware control functions are programmed into drivers that may have their own drivers or functions controlling the hardware.

This operating system and its drivers have been designed from the ground up to be resource efficient, this means low RAM and ROM usage. The scheduler itself makes use of processor sleep modes when possible.

4.3 Scheduler

The task scheduler used in OsasOS is a simple periodic task scheduler. This scheduler executes the tasks from a fixed task list. The tasks run to completion and are periodically executed, they keep their state in static structures or variables.

The task list contains a list of what tasks to execute, how often they are executed and what priority they have.

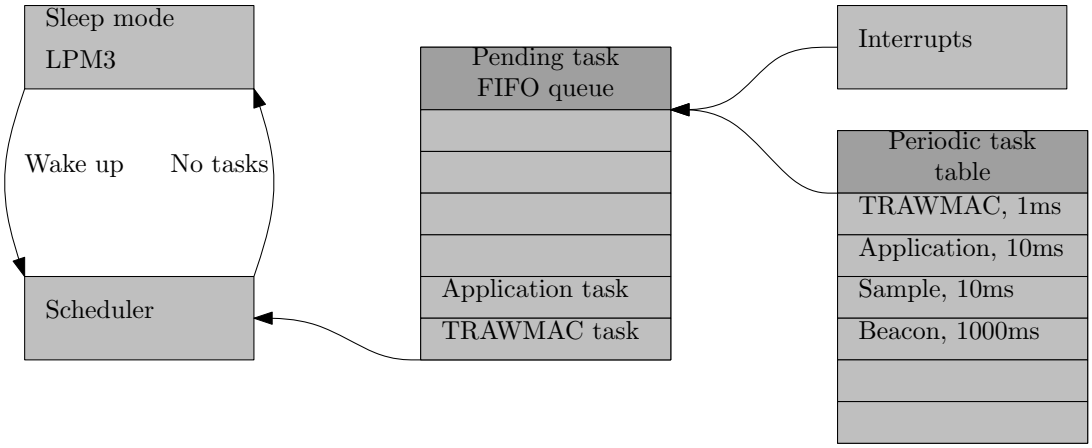


Figure 4.1: *Scheduling within OsasOS. Tasks are generated by the periodic task list and some sporadic tasks are generated by interrupts. These pending tasks are placed in a pending task FIFO queue. Interrupts (the periodic task list is also interrupt driven) wake up the CPU and runs the task execution loop. This loop checks the queue for any pending tasks and then executes these. When the queue is empty, the processor is put back to sleep.*

Each task is a standard C function that is periodically executed by the scheduler. The C function stores state and data in static variables that are preserved across function calls. The scheduler does not perform preemption but queues activations of tasks in a task queue.

The choice for using a non preemptive scheduler is done to conserve memory resources, especially RAM resources. As in a preemptive system multiple application stacks need to be stored in the RAM of the controller, and in the current nodes the RAM is a very limited resource.

4.3.1 Scheduling API

Type	Name	Description
volatile uint16_t	act_time	How long until activation of the task
uint16_t	act_interval	Activation interval of the task
volatile uint8_t	act_pend	Task activations pending
int8_t	act_done	Activations done
void (*) (void)	task_func	Function pointer to the task

Table 4.1: *Task structure*

The scheduler is a list of task entries that is evaluated by the scheduler to see what tasks need to be run. Each task entry consists of a task structure as seen in table 4.1. The task structure contains data of how often a task runs in milliseconds, how long until it runs, how many activations are pending and what function to run when the time arises.

The source code in program 1 shows how the task list is initialized, this is done at the

Program 1 Initialization of the task list

```
// Trawmac table updating task
tasks[0].act_time = tasks[0].act_interval = 0;
tasks[0].act_pend = tasks[0].act_done = 0;
tasks[0].task_func = TRAWMAC_task;
   // test task for led blinking
tasks[1].act_time = tasks[1].act_interval = 1000+5;
tasks[1].act_pend = tasks[1].act_done = 0;
tasks[1].task_func = blink;
   // beacon task
tasks[2].act_time = tasks[2].act_interval = (5000)-1;
tasks[2].act_pend = tasks[2].act_done = 0;
tasks[2].task_func = TRAWMAC_send_beacon;
// NULL task to show end of tasks
tasks[3].task_func = NULL;
```

start of running the runtime and operating systems. The array index indicates the priority of the task in the task list. The list should be terminated with a task that has a NULL pointer for the function to be executed.

4.3.2 scheduler operation

The physical implementation of the scheduler is implemented in three program blocks.

The scheduling loop is periodically called by the timer module at a regular interval of approximately one millisecond. The call can occur anywhere in the program and will suspend current program operation.

The scheduling loop decrements the timeout counters in use by the program. The timeout counters are an array of values that are counted down during each call of the scheduler routine. When the counters reach zero, the value is not decremented further.

The scheduler routine decrements all the activation counters of the task entry. If activation counter has reached zero, the task pending activation variable *act_pend* is incremented and the global task pending variable *global_pending*. After the counters have been updated, the task counter is reset.

The scheduling loop checks in the table if any tasks need to be executed. First a variable is checked if any tasks are pending. If not, the CPU will go to sleep. As there is work needed to be done, it will check the entries of the task list if any tasks are pending. Each time an task is executed the counters are incremented and the loop is exited. Then another check is done if any tasks are executed.

The scheduler is called from the main loop of the program. When the CPU is woken bij the timer interrupt, the scheduler is executed and checks are done to see if there any pending executions.

Program 2 Scheduling loop of the main program

```
// Timer A0 interrupt service routine
interrupt (TIMERA0_VECTOR) Timer_A(void)
{
  uint8_t i;
  // timer tick functionality
  ticks++;
  // decrement timeout counters
  for(i=0;i<MAXTASKS;i++)
    if(timeouts[i] > 0)
      timeouts[i]--;
  // decrement counters and activation periods.
  for(i=0;tasks[i].task_func != NULL;i++)
  {
    // check if the activation counter needs to
    // be counted down
    if(tasks[i].act_time != 0)
      // decrement activation timer
      tasks[i].act_time--;
    else
    {
      // reset activation timer
      tasks[i].act_time = tasks[i].act_interval;
      // add one more activation
      tasks[i].act_pend++;
      // add one to global
      global_pending++;
    }
  }
  neighbor_table_update();
  // exit LPM3 mode
  _BIC_SR_IRQ(LPM3_bits); // Clear LPM3 bits from 0(SR)
}
```

4.3.3 Example application, blink

To demonstrate how the OsasOS operating system functions we have created a small example called *Blink*. The blink task calls a function to toggle a LED on or off. The source code for the blink task is in program 5.

The task list for the blink application is given in program 7. The task list has two entries in total. One entry for the blink application with an execution interval of one second. The second entry is the terminating entry that ends the task list.

Program 3 Task execution loop.

```
// main scheduling loop
void schedule(void)
{
  uint8_t i;
  while(global_pending != global_done)
  {
    for(i=0;tasks[i].task_func != NULL;i++)
    {
      // work in this entry?
      if(tasks[i].act_pend != tasks[i].act_done)
      {
        // execute
        tasks[i].task_func();
        // increment done counters
        tasks[i].act_done++;
        global_done++;
        // recheck if work is done
        break;
      }
    }
  }
}
```

Program 4 Scheduler is called by the main loop, and then the CPU is put to sleep.

```
while(1)
{
  _BIS_SR(LPM3_bits);   // Sleep CPU
  schedule();           // wake up, schedule any pending tasks
}
```

Program 5 *Blink task code*

```
void blink(void)
{
  // toggle a led
  LED_toggle(LED0);
}
```

Program 6 *Blink task list*

```
// blink task information
tasks[0].act_time = tasks[0].act_interval = 1000;
tasks[0].act_pend = tasks[0].act_done = 0;
tasks[0].task_func = blink_task;
// end of task list
tasks[1].task_func = NULL;
```

4.4 Drivers

The operating system provides task scheduling and activation but also some control of the hardware is required. This is done by creating drivers for the relevant hardware parts of the nodes. There are drivers for the radio, ADC and some power control of the peripherals.

4.4.1 Example driver ADC

Drivers do not have a strict makeup of routines but do have a set of functions that are present in all drivers.

The ADC driver contains a few functions to control the ADC of the MSP430 microcontroller. Most of the functions set the hardware registers of the ADC peripheral to perform a certain function. For specific information on the registers see the datasheet of the microcontroller [Ins04].

The initialization function *void ADC_Init(void)* sets up the ADC converter. The ADC pins are switched from general I/O to analog input pins. The ADC clock options are setup, this determines sample rate and sampling accuracy. The ADC mode is setup, in this case it is setup to single conversion mode.

The *void ADC_off(void)* function turns the ADC off. To enable the ADC again, the initialization function should be called again.

The *uint16_t ADC_convert(uint8_t input)* function performs a single conversion on the pin defined by the argument *input*. The result of this conversion is returned back to the calling application.

4.4.2 Radio

The Radio driver controls the CC2420 [Ins07] transceiver on the BSN node. The radio interfaces with the transceiver by using the SPI bus of the MSP430F149 microcontroller. This bus is used to send commands to the transceiver, read and write data and query the transceiver state.

The transceiver has a state machine controlling the behavior of the radio and is controlled by events on the wireless medium or by commands sent by the microcontroller. These commands are also called *strobes* in the CC2420 documentation [Ins07] and provide fast method in switching the operational state of the radio. Example command strobes are resetting the transceiver, starting reception mode, switching to a power mode. Each strobe command the radio also outputs a state value that contains the current state, the microcontroller can respond to the possible error states. The strobe functionality is partially exposed to provide

Program 7 *ADC driver code*

```
// initialize ADC converter
void ADC_Init(void)
{
  // Set all ADC pins to active
  P6SEL |= 0xFF;
  // setup ADC clock and buffering
  ADC12CTL0 = ADC12ON + SHT0_2;
  // set ADC to single conversion mode
  ADC12CTL1 = SHP;
}
// Turn off ADC converter
void ADC_off(void)
{
  // turn ADC off
  ADC12CTL0 &= ~ADC12ON;
}

// Start ADC conversion on channel input
uint16_t ADC_convert(uint8_t input)
{
  // change channel
  ADC12MCTL0 = input;
  // enable conversion
  ADC12CTL0 |= ENC;
  // start conversion
  ADC12CTL0 |= ADC12SC;
  // wait until done
  while((ADC12IFG & BIT0) == 0)
    ;
  // clear conversion start
  ADC12CTL0 &= ~ENC;
  // return value to application
  return ADC12MEM0;
}
```

radio sleeping, power down and start receiving data.

The transceiver also contains data queues for transmission and reception of packets. The state of these queues is described in what state the radio is and in a few dedicated outputs on the radio. The transmit queue holds one single packet and is transmitted when the command is given. The reception queue can hold multiple packets and can be read out when other packets are receiving.

The driver responds to the state of the radio and changes on the queue state outputs of the radio by retrieving the data from the reception queue. The size of the packet is known

by reading out the first byte of the packet and then an array is allocated with the size of the packet. This allocated packet is checked and if the packet data is all in order (checksum, addresses) the packet is placed in the reception queue to be analyzed by the MAC layer.

4.4.3 Microcontroller

The microcontroller contains some peripherals that have simple wrapper functions. The ADC has an initialization function that sets up the clocking and buffering correctly for the ADC. There is a function to set the channel, this selects what pin the ADC will measure. Then there is a single conversion mode where the ADC will perform a single AD conversion. Most of the sensors on the BSN node sensor boards are connected to a few pins that can be sampled by the ADC. The SPI bus is used to transfer data to and from a few devices on the BSN node, this includes the radio and the serial flash.

4.4.4 Peripherals

The BSN node contains some on board hardware next the radio. There is a 512KiB flash chip present for mass storage, this chip can be accessed via the SPI bus. Reading and writing to the serial chip is currently not implemented. Only the command to power down can be given to turn the serial flash in a lower power mode. There is also a small set of wrapper functions to turn on and off the radio, serial flash and the sensor board. This way power consumption can be regulated to a bare minimum, and devices not used can be turned on and off. There are some pitfalls with this approach, especially with the serial flash. When the serial flash is not powered the chip will load down the serial bus and cause communication problems.

Chapter 5

Experiments

Now that a working new platform has been created by combining OsasOS with TRAWMAC and the OSAS runtime we can see how much improvements have we gotten.

We will measure the improvement in resource usage of the new software stack compared with other stacks, and the differences in power consumption.

5.1 setup

We are interested in the improvements in power consumption and resource consumption of the new software stack. We use the static output of the compiler toolchain to measure the resource consumption like ROM and RAM memory. For measuring power consumption we use a dedicated energy measurement device.

5.1.1 Resource measurement

Resource consumption is measured through using a tool from the GNU binutils package known as *msp430-size*. By calling this tool with the ELF file of the firmware we can see how much ROM and statically allocated RAM is used.

```
msp430-size trawmac_tester.elf
text    data     bss     dec     hex filename
7760       8     372    8140    1fcc trawmac_tester.elf
```

The text field shows how much flash memory is used while the data and bss fields show how much RAM is used.

This measurement is not entirely fair as we have used a lot of dynamic memory allocation. This does improve the memory utilization as in the previous software stack a few large static buffers where used. These buffers where always maximum size, regardless of content.

5.1.2 Energy measurement

We need to measure energy consumption, this can be done by counting the amount of cycles of the CPU and the on time of the wireless chipset, but they all need to be modeled and characterized. Instead we chose to measure power consumption by electronic means with the aid of a Coulomb Counter. This coulomb counter sits in between the battery pack and the sensor node itself, see figure 5.1.

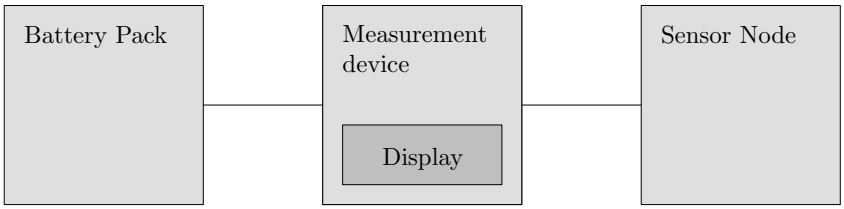


Figure 5.1: *Schematic of the location of the energy measurement device. The device itself sits in between the user of power and a battery pack. The current flowing trough the device is measured over time.*

Coulomb counter

The Coulomb counter is a device that is typically used in energy metering. One classic example of a coulomb counter is the electricity meter in your house. It measures the amount of power flowing through the net of your house and counts the amount of watt hours consumed.

$$U = I * R \tag{5.1}$$

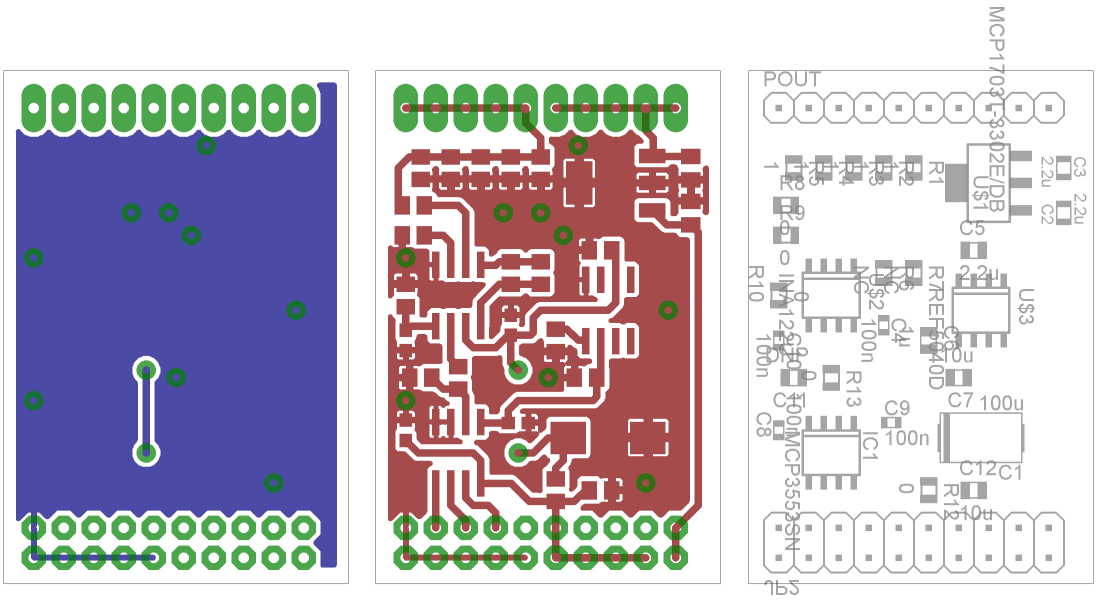
The coulomb counter we use in our experimental setup is similar in operation but has no mechanical components but measures the current by electronic means. This is achieved by letting the unknown current flow through a small but precise resistor and measuring the voltage difference across the resistor. By using Ohm's law in the test (Equation 5.1) we can calculate the power consumed at that instant. The electronic term for this "sensing" resistor is also called a *Shunt resistor*.

As the shunt resistor used is small, so is the voltage small that falls across the it. This small voltage is not suitable to get directly measured, so a amplifier is used. There are specialized amplifiers that can amplify small voltages and output the difference between two inputs, these are called *instrumentation amplifier*. The amplifier we use in our hardware is the INA122 [Cor97] low power, single supply instrumentation amplifier from Texas instruments. This amplifier outputs the difference voltage between the two inputs, referenced to a reference voltage.

The voltage reference used in the measurement hardware outputs 2.048 Volts and is based around the REF5040 [Inc] high precision, low drift reference from Texas instruments.

The amplifier, shunt and reference circuit output a voltage of 2.048 at no load, and this voltage decreases until 0 volts with the maximum load of 50 milliamperes.

There is one problem with this particular setup. Digital circuits like our wireless sensor nodes have irregular power consumption. When there is no switching activity the node does consume barely any power, but when active large current consumption spikes occur. Measuring the current to also take into account these spikes is infeasible as that would increase complexity. We have chosen to create a low pass filter with the node itself to filter out the large current consumption spikes, this is achieved by a large capacitors that will smooth these spikes into average current consumption.



(a) PCB bottom layer **(b)** PCB top layer **(c)** PCB part layout

Figure 5.2: *Circuit board layout of the analog section. Contains the amplifier, reference, ADC and 3.3V regulator for the wireless sensor.*

The digital section is built around the ATmega16 [Cor09b] Microcontroller from Atmel. The voltage from the analog section is fed into a 22bit ADC MCP3553 [Cor09c] (Analog to Digital Converter) that is connected via an SPI bus (Serial Peripheral Interface) to the microcontroller. The microcontroller performs a integration over time to calculate and store the average power consumption. The limited sample rate of the ADC is compensated for by using a sufficiently large bypass capacitor that will filter out any sudden changes in power consumption.

Coulomb counter usage

The coulomb counter has a simple user interface consisting of a character LCD screen and four buttons.

The character LCD screen shows the following values: The Current measurement for this second labeled with **I**. The Zero offset value labeled with **Z**. The Current coulomb recorded labeled with **C**. The amount of seconds running labeled with **t**. There are also some buttons present:

Start Start measuring, before connecting a new device or hardware, wait about 20 seconds for the supply to stabilize.

Pause Pause measurement, ideal for noting measurements.

Zero Transfer current average power consumption to the Zero variable, this number is subtracted from the measurements. Do not forget to reset afterwards.

Reset Clears the current measurement except for the Zero value.

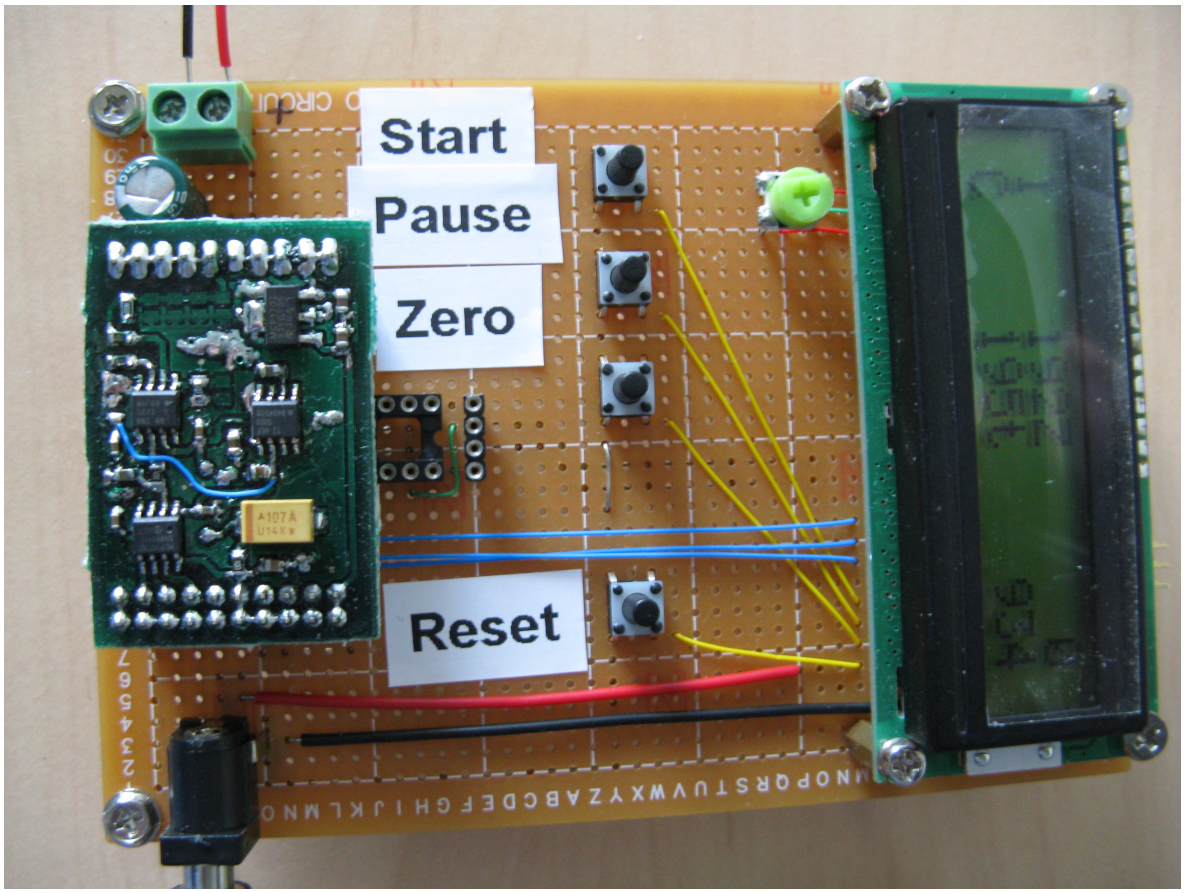


Figure 5.3: *Digital section with display and buttons*

Before starting with measurements a calibration step is necessary to remove any imperfections in the analog circuitry. The is done by disconnecting any loads of the coulomb counter and then performing a measurement of 10 minutes. this measurement shows the offset measurement that is introduced by the imperfections of the analog circuitry. This value can be stored as an offset by using the *Zero* button.

5.2 Measurements

Now that we have a broad overview what we want to measure and that we have the hardware/software to do so, we will go into detail of what we want to measure.

5.2.1 WSN hardware

We want to measure how much power the hardware of the wireless sensor node consumes and also how the power consumption is influenced under various power modes. This information is useful not directly for the thesis research but also for application developers.

We focus primarily on power consumption measurements of the radio and the CPU. The radio has the highest power consumption of all the components and has various operational

modes. The processor only plays a small part in power consumption but as the node sits in sleep mode often, in the long run it influences node life time.

Most of the data measured here is also noted in the datasheet of the device but these are often worst case power consumption or typical power consumption. There are also external factors that can make a calculated estimate of the power consumption optimistic or pessimistic.

The MSP430F149 [Ins04] is a microcontroller optimized for low power operation and has various sleep modes that each has their own power consumption and properties, see table 1.1. We intend to measure the power consumption of used operational modes of the processor and some computation operations.

The CC2420 [Ins07] has various modes with varying power consumption and properties. There is a large flexibility in selecting the transmission power of the radio so this is useful to investigate when implementing a MAC protocol that can exploit these settings.

Hardware experiments planned.

- CPU in sleep mode with timer operational, interrupt service routine disabled.
- CPU in sleep mode with timer operational, interrupt service routine active but empty.
- CPU in sleep mode with scheduler operational and receiving.
- CPU in sleep mode with scheduler operational and transmitting once every second at various transmission powers.

5.2.2 operational measurements

Next to the pure hardware energy consumption measurements we need to measure how the software performs on the wireless sensor nodes and what influence various software modes have on the power consumption.

We want to measure how much power the operating system consumes while doing various tasks. The MAC protocol has various operational mode and we want to investigate how much power it consumes in various operational modes.

- OsasOS Scheduler operational with zero tasks
- OsasOS Scheduler operational with one dummy task
- OsasOS scheduler operational with one task with some work.
- OsasOS scheduler operational with one analog sample per second.
- OsasOS scheduler operational with ten analog samples per second.
- OsasOS scheduler operational with 100 analog samples per second.
- TRAWMAC 10 second interval.
- TRAWMAC 1 second interval.
- TRAWMAC 0.1 second interval.

Next to just operational characteristics we will also see how the new software stack compares to present stacks like TinyOS 1.0 and the mantisOS stack.

Chapter 6

Results

We have conducted the measurements outlined in chapter 5 and present the numbers here.

6.1 Baseline

Device	Conditions	measured	expected	deviation
$1K\Omega$ resistor 1%	3.3V, 20 °C	3228μA	3300μA	2%
$10K\Omega$ resistor 1%	3.3V, 20 °C	324μA	330μA	2%
$100K\Omega$ resistor 1%	3.3V, 20 °C	32μA	33μA	3%
$1000K\Omega$ resistor 1%	3.3V, 20 °C	3μA	3.3μA	9%

Table 6.1: *Calibration measurements of the energy measurement device. The low accuracy at the lowest level is due to lack of resolution from the measurement device.*

Not noted in the Experiments chapter but included for clarity are measurements of the accuracy of the energy meter. These are static measurements to verify the accuracy of the energy measurement device. Before measurement with one resistor is made an offset calibration is done. This offset calibration is done to remove any constant offsets in the measurement device. This offset originates from the imperfections in the energy measurement device.

The accuracy across the various consumption rates varies a few percent except for the lowest power consumption range. The overal accuracy is acceptable, no real figure was given for wanted accuracy but 5% was acceptable. The equipment exceeds this except for the lowest value. The lowest value power consumption is hardest to measure as the signal is approaching resolution and noise floor of the amplifier and ADC. This could be improved by using an ADC with higher resolution and a lower noise amplifier.

6.2 WSN hardware

Table 6.2 shows the measurements of the wireless sensor hardware: Processor running in various modes and loads, Radio in various modes, sensor power consumption. Most of the observations match the typical data consumption noted in the datasheet of the radio and processor. The sensor board had insufficient documentation about its operational characteristics so we did not have a proper estimate of the power consumption before starting measurements.

Device	Conditions	Current
Processor LPM 3, timer active, no interrupts	3.3V, 20 °C	64μA
Processor LPM 3, timer active, empty timer interrupt	3.3V, 20 °C	48μA
Processor LPM 3, timer active, empty timer interrupt, sensor board on	3.3V, 20 °C	640μA
Processor LPM 3, scheduler active, no tasks	3.3V, 20 °C	207μA
Processor LPM 3, scheduler active, 1 empty task activated 1024 times per second	3.3V, 20 °C	246μA
Processor LPM 3, scheduler active no tasks, radio idle	3.3V, 20 °C	634μA
Processor LPM 3, scheduler 1 radio task every 128 ticks, radio receiving	3.3V, 20 °C	18984μA
Processor LPM 3, scheduler 1 radio task every 64 ticks, radio Transmit at 0dB 64 bytes 16 packets per second	3.3V, 20 °C	4213μA
Processor LPM 3, scheduler 1 radio task every 64 ticks, radio Transmit at -10dB 64 bytes 16 packets per second	3.3V, 20 °C	2420μA
Sampling sensors at 100Hz	3.3V, 20 °C	1618μA

Table 6.2: *Power consumption of wireless sensor peripherals.*

One interesting note is the first two measurements in the table. The MSP430 consumes more power when no interrupts are handled then when they are handled. We have no explanation for this, the datasheet does not mention anything. We do have a theory that somehow that when interrupts are pending and not handled the controller consumes more power. This is not a normal condition during operation so is of no influence in later tests.

6.3 operational measurements

Device	Conditions	Current
TRAWMAC, 125ms, transmit 7 bytes	3.3V, 20 °C	14.5 milliampere
TRAWMAC, 125ms, receive 7 bytes	3.3V, 20 °C	11.9 milliampere
TRAWMAC, 1s, transmit 7 bytes	3.3V, 20 °C	3.9 milliampere
TRAWMAC, 1s, receive 7 bytes	3.3V, 20 °C	2.2 milliampere
TRAWMAC, 10s, transmit 7 bytes	3.3V, 20 °C	1.2 milliampere
TRAWMAC, 10s, receive 7 bytes	3.3V, 20 °C	1.1 milliampere
TRAWMAC, 30s, transmit 7 bytes	3.3V, 20 °C	1.1 milliampere
TRAWMAC, 30s, receive 7 bytes	3.3V, 20 °C	1.0 milliampere
Sampling 100Hz, TRAWMAC transmit 18byte per 250 msec	3.3V, 20 °C	12477μA
Sampling 100Hz, TRAWMAC transmit 18byte per second	3.3V, 20 °C	4275μA
Sampling 100Hz, TRAWMAC transmit 18byte per 5 seconds	3.3V, 20 °C	2225μA

Table 6.3: *Power consumption of some operations like those from the MAC layer and sensor readings*

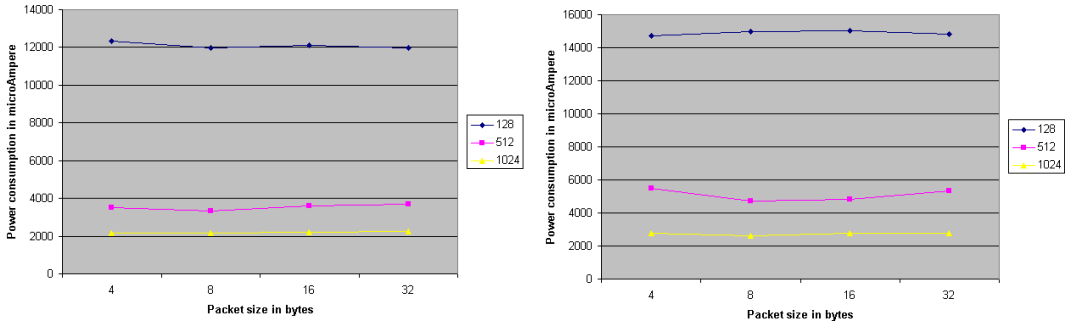
Table 6.4 shows how the TRAWMAC protocol performs during operation in various transmission intervals, the measurements for transmitter and receiver have been included. Next to values with only TRAWMAC measurements there are also some operational tests where

sampling occurs with some periodic transmission of data. This is a approximation of a real life application test where the node is transmitting samples at a regular interval.

6.4 TRAWMAC detailed

	Size in bytes			
CI	4	8	16	32
128	TX:14702 RX:12341	TX:14962 RX:11973	TX:15039 RX:12122	TX:14827 RX:11989
512	TX:5495 RX:3520	TX:4705 RX:3345	TX:4836 RX:3598	TX:5318 RX:3681
1024	TX:2792 RX:2781	TX:2638 RX:2143	TX:2773 RX:2192	TX:2784 RX:2257

Table 6.4: *TRAWMAC operational measurements, first number is the transmission power consumption, second number is reception power consumption. The values are in micro ampere's and are measured with a 3.3V power supply at temperature of 20 degrees centigrade.*



(a) Reception **(b)** Transmit

Figure 6.1: *Power consumption graph, using the values of table 6.4*

Next to general measurements with various operations, we have measured few check intervals and packet sizes. The results are put in table 6.4 and plotted in the graphs in figure 6.1.

The values for the packet size make little sense as some larger packets take less energy then smaller packets. This is due to an error in the test case that was not noticed right away. The graphs do show how increased CI's improve power consumption, it is expected that larger packets consume more to transmit but this should be small percentage.

6.5 WSN stacks

The other WSN stacks have much higher power consumption, and they draw this power regardless of activity on the radio or processor (table 6.5). What is a bit perplexing is the power consumption is higher then the maximum power consumption of all the devices in the wireless sensor node, this is for TinyOS and MantisOS. We have contacted some of the developers and there seemed to be a problem with power gating control. There is also a revised

Device	Conditions	Current
TinyOS	3.3V approx 20 degrees centigrade	42mAmpere (out of range)
MantisOS	3.3V approx 20 degrees centigrade	38mAmpere
MantisOS and WISEMAC	3.3V approx 20 degrees centigrade	4.5mAmpere

Table 6.5: *Power consumption of other wireless sensor network stacks.*

measurement of MantisOS with WISEMAC that seems to be more realistic. The MantisOS and WISEMAC stack comes close to the OsasOS and TRAWMAC implementation. The advantage of TRAWMAC here is that power consumption can be reduced when there is not a lot of traffic present.

6.6 Resource usage

	MantisOS	TinyOS	OsasOS with TRAWMAC
RAM	1.1KiB	1.3KiB	0.6KiB
ROM	32KiB	27KiB	22KiB

Table 6.6: *Comparing resource usage between the OsasOS and TRAWMAC stack with MantisOS and TinyOS*

Next to power usage optimization we also measured the resource usage of the system itself compared to two existing software stacks, see table 6.6. The resource consumption is reduced with OsasOS and TRAWMAC, especially in RAM usage. This is due to usage of dynamic memory allocation for the buffers and the zero copy semantics used in transferring the buffers to different layers within the system.

6.7 Energy model

By reading the data from the datasheet and seeing how the TRAWMAC protocol behaves we have created a model that is suitable for use in simulation tools.

Radio mode	Symbol	Current
PowerDown	$A_{powerdown}$	20μA
Idle	A_{Idle}	426μA
Receive	A_{RX}	18.8mA
Transmit	A_{TX}	17.4mA

Table 6.7: *Power consumption figures of each radio mode used by the TRAWMAC energy model. These values are taken from the datasheet [Ins07] and assume an supply voltage of 3.3V.*

$$C_{period} = t_{mode} * A_{mode} \tag{6.1}$$

$$C_{CI} = \sum_{i=0}^{n} C_{mode_i} \tag{6.2}$$

TRAWMAC action	Symbol	Time duration	radio mode
Check interval Idle	t_{IDLE}	$t_{CI} - \sum_{i=0}^{n} t_{modes}$	A_{Idle}
CCA	t_{CCA}	32milliseconds	A_{RX}
transmit MFP	$t_{\text{MFP transmit}}$	2milliseconds	A_{TX}
transmit MFP acknowledge	$t_{\text{MFP ACK transmit}}$	2milliseconds	A_{TX}
Receive MFP acknowledge	$t_{\text{MFP ACK receive}}$	32milliseconds	A_{RX}
transmit data< 32 bytes	$t_{\text{Data transmit}}$	2milliseconds	A_{TX}
transmit data> 32 bytes	$t_{\text{Data transmit}}$	3milliseconds	A_{TX}
Receive data	$t_{\text{data receive}}$	32milliseconds	A_{RX}
transmit data acknowledge	$t_{\text{data ACK transmit}}$	2milliseconds	A_{TX}
Receive data acknowledge	$t_{\text{data ACK receive}}$	32milliseconds	A_{RX}

Table 6.8: *Time durations of various TRAWMAC actions.*

$$A_{CI} = C_{CI}/t_{CI} \tag{6.3}$$

$$W_{CI} = A_{CI}/V_{system} \tag{6.4}$$

In our hardware the active power of the radio composes the largest part of the power consumption of the node. In our energy model we divide up the power consumption of the radio in four modes. The TRAWMAC protocol only utilizes three modes. The power consumption of the modes is described in table 6.7.

The TRAWMAC protocol spends a amount of time in the radio modes. We need to determine how much time is spent in each mode. There are various actions in the protocol that each have their own duration. The actions in one communication also differ for the transmitter and receiver side. These durations are stored in table 6.8.

The TRAWMAC protocol performs a sequence actions for each communication. The protocol description describes what actions are performed for each communication, see figure 2.4. The diagram shows what modes the receiver and the transmitter enter for communication.

To compute the total amount of power consumed we add together the power consumption of each of the communications actions. Equation 6.1 shows how the time period length t_{mode} is multiplied with the power consumption of the radio mode A_{mode}. The result of this equation is given in coulombs.

The sum of all mode changes and periods is combined into the total sum of coulomb consumed during the CI. This is done in equation 6.2

Now that the total coulomb usage is known, we can convert this back to the current consumed over one CI by using equation 6.3. If further conversion to watts is desired, the current consumed over the CI A_{CI} is multiplied by the supply voltage V_{system} (see equation 6.4).

6.8 Conclusions

The TRAWMAC protocol proves to be a good improvement over the current software stacks as can be seen from the measurements. The other WSN stacks continually use power while even at high load TRAWMAC proves to be a better solution.

As was expected, increasing the CI's reduces power consumption as the duty cycle of the radio is decreased. The power consumption is asymmetric with respect to transmitter and receiver. The transmitter initiates the communications and is active for a longer period then the receiver.

There is a point where the overhead of the protocol proves to be a problem. At high loads the overhead of the MFP packets and acknowledgements will create large amounts of traffic, introducing packetloss. The reason for this is that multiple packets going to the same destination are each transmitted individually and not in a stream of packets. Another problem is the response time of the radio itself, various actions need to be performed before a transmission can be performed, this increases the latency by a few milliseconds.

Creating a system from the ground while keeping in mind to conserve power and resources seems to be a good approach of creating a suitable software stack for wireless sensors networks. During the development of OsasOS, some intermediate power measurements where performed to verify if the low power modes of the CPU where properly exploited, the same was done for the radio operation modes.

Having a device that measures power consumption is useful to verify if the software is operating normally. This can be used to check if the operating system is behaving normally and to verify if a MAC under development is behaving as expected. When creating larger networks of wireless sensors the measurement of all the nodes becomes hard to do as many measurement devices are required, a model based approach scales better in the end verification.

6.9 Improvements

The software stack as it is needs some improvements for more optimal power efficiency or better software support.

The TRAWMAC protocol defines shortened data communications in the form of DFP packets. In the current implementation there is no support for DFP packets. DFP's reduce overhead when transmitting small amounts of data as the DFP communications skips a part of the communication handshaking and acknowledges.

In the current implementation when data is ready for transmission, the radio is started right away. If the data have no immediate deadline then data could also be deferred after a CCA is performed, the radio is already in an awake state and can be used right away when needed.

When using long CI's and there is no data pending for a while, the radio can be best put into deep sleep mode and save a lot more power. Currently the radio is continually in stand by mode and consumes about $426\mu Ampere$ current while power down mode consumes $20\mu Ampere$. This optimization would reduce power consumption even more.

Neighborhood tables can be useful information to transmit to other neighbors. New nodes added to the network can get updates about their neighborhood without overhearing them first.

Currently Neighborhood entries always remain in the neighborhood table list, even after neighbors drop out of range. A change is required here to remove neighbors from the table by using a least recently used removal policy.

Most functions and queueing of the MAC API have not been implemented, this is not a large programming effort and the TRAWMAC protocol has all the provisions to be coupled

to the API.

The operating system needs a more efficient scheduler. The task table is now a list with counters that decrements periodically. This could use a better implementation.

Nodes can potentially wake up at the same period, this will generate collisions if both of are targets of communication. There needs to be a mechanism where one node can detect its schedule overlaps with that of a neighbor. It should move itself into a new slot.

6.10 Future work

The TRAWMAC protocol can be improved by using a different radio chip. The CC2500/CC1100 has as a useful feature that they can periodically wake up, sample the medium and go back to sleep automatically. This would save idle power consumption greatly as the radio is in deep sleep mode the whole time, consuming minimal current, This is called by the manufacturer *Wake-on-radio*. This feature can be exploited by preamble sampling MAC protocols like TRAWMAC to improve sleeping power consumption. The interval where Wake-on-radio is performed can be changed as can the sampling period.

The CC2420 does not contain any useful information about duration of the state transitions, the CC2500/CC1100 do have an detailed table with absolute maximums of durations. Such information is very useful for fine tuning TRAWMAC to reduce the on time and improve synchronization. This can be even more improved if the MAC has bit level control of the medium, this does incur a higher processing overhead but does improve flexibility and improve preamble sampling support.

Making the MAC more efficient and compact enough that it is suitable to place the MAC statemachine into the interrupt service routine. This would greatly improve reaction latency and combined with an radio that has deterministic timing would improve efficiency and throughput.

Bibliography

[AW06] T. Doan Q. Cao L. Selavo Y. Wu L. Fang Z. He S. Lin J. Stankovic A. Wood, G. Virone. Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Technical report, Department of Computer Science University of Virginia, 2006.

[BLV09] R. Bosman, J. Lukkien, and R. Verhoeven. An integral approach to programming sensor networks. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5, Jan. 2009.

[CG01] Jan. M. Rabaey Chunlong Guo, Lizhi Charlie Zhong. Low power distributed mac for ad hoc sensor radio networks. 2001.

[con09] WASP consortium. The wasp project, http://www.wasp-project.org/, 2009. [Online; accessed 19-october-2009].

[Cor97] Burr-Brown Corporation. *INA122, Single Supply, MicroPower INSTRUMENTATION AMPLIFIER*, April 1997.

[Cor09a] Atmel Corporation. *4-megabit 2.5-volt or 2.7-volt DataFlash AT45DB041D*, March 2009.

[Cor09b] Atmel Corporation. *Microcontroller with 16K Bytes In-System Programmable Flash*, Oktober 2009.

[Cor09c] Microchip Corporation. *MCP3550/1/3, Low-Power, Single-Channel 22-Bit Delta-Sigma ADCs*, March 2009.

[CS03] Young-Bae Ko Changsu Suh. A traffic aware, energy efficient mac protocol for wireless sensor networks. 2003.

[DGV04a] Adam Dunkels, Bjrn Grnvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[DGV04b] Adam Dunkels, Bjrn Grnvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[EHD04] A. El-Hoiydi and J.-D. Decotignie. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In *ISCC '04: Proceedings of the Ninth International Symposium on Computers and Communications 2004 Volume 2 (ISCC"04)*, pages 244–251, Washington, DC, USA, 2004. IEEE Computer Society.

[HB07] M. Hefeeda and M. Bagheri. Wireless sensor networks for early detection of forest fires. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–6, Oct. 2007.

[Inc] Texas Instruments Incorporated. *REF5040, Low-Noise, Very Low Drift, Precision VOLTAGE REFERENCE.*

[Ins04] Texas Instruments. *MSP430x13x, MSP430x14x, MSP430x14x1 Mixed Signal Microcontroller*, June 2004.

[Ins07] Texas Instruments. *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, March 2007.

[JAM08] Xi Zhang Junaid Ansari and Petri Mahonen. Poster abstract: Traffic aware medium access control protocol for wireless sensor networks. Technical report, Department of Wireless Networks, RWTH Aachen University, Kackertstrasse 9, D-52072, Aachen, Germany, 2008.

[JP04] David Culler Joseph Polastre, Jason Hill. Versatile low power media access for wireless sensor networks. 2004.

[KA03] Mohamed Younis Kemal Akkaya. A survey on routing protocols for wireless sensor networks. 2003.

[Lan07] K. Langendoen. Medium access control in wireless sensor networks. In H. Wu and Y. Pan, editors, *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*. Nova Science Publishers, Inc., 2007.

[LC02] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. 2002.

[Lon] ICL London. *http://ubimon.doc.ic.ac.uk/bsn/m621.html.*

[MFHH05] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[ND99] Asis Nasipuri and Samir R. Das. On-demand multipath routing for mobile ad hoc networks. 1999.

[NX04] Krishna Kant Chintalapudi Deepak Ganesan Alan Broad Ramesh Govindan Deborah Estrin Ning Xu, Sumit Rangwala. A wireless sensor network for structural monitoring. Technical report, 2004.

[PLC03] J. Polastre R. Szewczyk K. Whitehouse A. Woo D. Gay J. Hill M. Welsh E. Brewer P. Levis, S. Madden and D. Culler. Tinyos: An operating system for sensor networks. 2003.

[RGG05] Omprakash Gnawali Ramakrishna Gummadi and Ramesh Govindan. Macro-programmingwireless sensor networks using kairos. 2005.

[RGLAO05] V. Rajendran, J. Garcia-Luna-Aceves, and K. Obraczka. Energy-efficient, application-aware medium access for sensor networks. In *2nd IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2005)*, Washington, DC, November 2005.

[SBH05] HUI DAI JING DENG JEFF ROSE ANMOL SHETH BRIAN SHUCKER CHARLES GRUENWALD ADAM TORGERSON SHAH BHATTI, JAMES CARLSON and RICHARD HAN. Mantis os: An embedded multithreaded operating system forwireless micro sensor platforms. 2005.

[SCV03] Anuj Puri Sinem Coleri and Pravin Varaiya. Pedamacs: Power efficient and delay aware medium access protocol for sensor networks. 2003.

[Tec] Crossbow Technologies. *Telos-B mote platform.*

[VR03] J.J. Garcia-Luna-Aceves Venkatesh Rajendran, Katia Obraczka. Energy-efficient, collision-free medium access control for wireless sensor networks. 2003.

[YHE02] Wei Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576 vol.3, 2002.