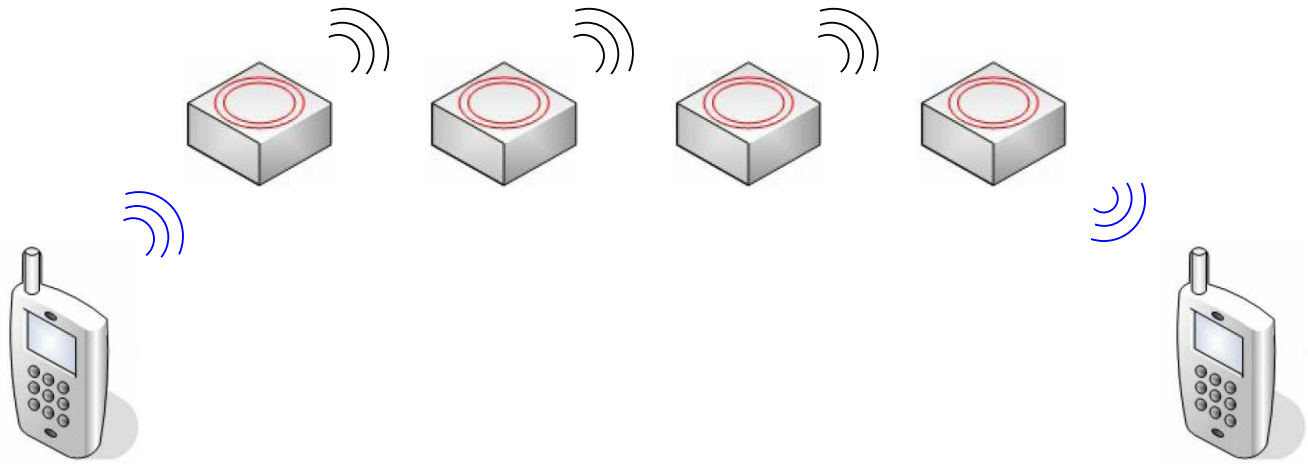
MAC Protocol for Wireless Sensor Networks

Group 652
Giuseppe Calí, Ileana Ghizdăvescu, Anders Grauballe, Mikkel Gade Jensen, Fabio Pozzo

Supervisors: Tatiana Kozlova Madsen, Frank H.P. Fitzek



6th Semester
Spring 2007
Institute for electronic systems
Communication systems

AALBORG UNIVERSITY

Department of Electronic Systems

Communication Systems

Fredrik Bajers Vej 7C

Telephone +45 96 35 87 00

Fax +45 98 15 17 39

http://www.kom.aau.dk

Title:

MAC protocol for wireless sensor networks

Theme:

Robust communication

Project period:

6th semester,
February - May 2007

Project group:

Communication Systems, group 652

Participants:

Giuseppe Calí
Ileana Ghizdăvescu
Anders Grauballe
Mikkel Gade Jensen
Fabio Pozzo

Supervisors:

Tatiana Kozlova Madsen
Frank Fitzek

Number of prints: 7

Number of pages: 95

Number of appendixes and character:

1 pcs. CD-ROM

Finished: May 30th, 2007

Synopsis:

Wireless sensor networks consist of spatially distributed autonomous small devices, often called "motes", which cooperatively monitor, collect and exchange data from the surrounding environment.
Aalborg University has developed a mote which integrates a Bluetooth module and a low cost Industrial, Scientific and Medical (ISM) band module which makes possible to establish a multi-hop connection between the motes.
The purpose of this project is to design and implement a Medium Access Control (MAC) protocol for the ISM module of the mote platform, proving a solution to avoid collisions between packets during transmission. A collision avoidance scheme with acknowledgements and carrier sensing has been designed and implemented to minimize data loss and duplication. This is also known as Request-To-Send (RTS) / Clear-To-Send (CTS) medium reservation mechanism.
To test the implementation of this protocol, a mobile phone application is developed which allows a user to exchange text, image and audio files through the mote network.
The acceptance test concludes that the implementation is robust and works as stated in the requirements specification except for one requirement regarding maximum transmission range. This is however due to the antenna calibration and is not software related.

Preface

This project has been carried out by project group 652, Communication Systems on 6th semester at Aalborg University, spring 2007. The focus group for this report is people with an interest in wireless sensor networks, MAC protocols, mobile development and the idea of communication between sensors and mobile phones.

In this report figures, pictures and tables are labeled with chapter and figure number for easy reference, e.g. 4.2 for second figure in chapter 4. References for literature are shown as e.g. [10].

A CD is attached to the back cover of the report containing the following:

- The report in PDF format.
- The source code for the programs in C and Python.

There are several people we would like to thank for their involvement and for helping us completing this project. Therefore, we would like to express our gratitude to our supervisors Tatiana Kozlova Madsen and Frank F. Fitzek who have guided us in our work. Their advice, idea and support, throughout the project has been very helpful. The same holds true for Ben Krøyer who designed and built the sensors. Furthermore, we are thankful to Stephan Rein and Daniel Gühne from TU Berlin for code examples and programming support. Finally, our thanks go to Gian Paolo Perrucci, for the Python programming support.

Giuseppe Calí	Ileana Ghizdăvescu
Anders Grauballe	Mikkel Gade Jensen
Fabio Pozzo	

Contents

1 Introduction **8**

1.1 First AAU mote 10

1.2 Second AAU mote 11

1.3 Initial problem 14

2 Analysis **15**

2.1 The data link layer 15

2.2 Multiple Access Protocols 17

2.3 Multiple access in wireless networks 22

2.4 Other sensor network issues 28

2.5 Problem statement 29

3 Requirements specification **30**

3.1 Preconditions 30

3.2 System requirements 30

3.3 Mote and protocol requirements 31

3.4 Mobile phone application requirements 31

4 System design **33**

4.1 Network scenario 33

4.2 Network interfaces 36

5 Mote protocol design **41**

5.1 Program design 41

5.2 UART/BT module 46

5.3 Timer module 48
5.4 Packet handling module 50
5.5 SPI/nRF module 51

6 Mobile application design 54
6.1 General description 54
6.2 Sender 56
6.3 Receiver 58
6.4 Graphical user interface 59
6.5 Modules 61

7 Implementation 63
7.1 Mote protocol 63
7.2 Mobile phone application 67

8 Test 69
8.1 Test of the mobile phone application 69
8.2 Test of the collision avoidance scheme 69
8.3 Acceptance test 70

9 Conclusion 73

10 Future perspectives 75

Bibliography 77

A Acceptance test specification 79

B nRF ShockBurst flow charts 86

C Bluetooth send/receive switching test 88

D Mobile phone application test 90

E Transmission range test 92

F Motes hardware schematics 93

Chapter 1

Introduction

The demand for gaining information about the surrounding environment is growing and throughout the last decades devices has been invented for this purpose. One of the new topics in this research area is wireless sensor networks that can provide this information in a fast and easy way. Wireless sensor networks consists of several small devices referred to as motes that in a cooperative way can collect and exchange information from the surrounding environment. They can be used to monitor e.g. temperature, pressure, motion etc. or they can be utilized to create a small wireless communication network. Wireless sensor networks benefits from other traditional networks in price and size. A qualitative comparison of wireless sensor networks, wireless ad-hoc networks and wired Networks can be seen in Table 1.1. The table can help deciding which kind of network to deploy in a given scenario.

Characteristic	**Wireless Sensor Networks**	**Wireless Adhoc networks**	**Wired Networks**
Transmission range	Short(3 to 30 m)	Comparatively longer(10 to 500 m)	Large(Up to 1 km)
Processing Power and Memory	Limited processing and memory capacity	Higher processing power memory	Highest available processing power and memory
Cost of nodes	Inexpensive	Relatively expensive nodes	Variable but expensive compared to wireless sensor nodes
Power supply	Nonrechargeable Irreplaceable	Rechargeable and/or replaceable batteries	Power outlets
Data rate	Low; 1-100 kb/s	High 54 Mbps	Highest 10-100 Mbps
Direction of flows	Predominantly unidirectional-sensor nodes to sink	Bidirectional end-to-end flows	Bidirectional end-to-end flows
Addressing	No globally unique ID	Globally unique ID	Globally unique ID

Table 1.1: *Comparison between Wireless Sensor Networks, Wireless Ad-hoc networks and Wired Networks [1].*

Figure 1.1 shows a possible setup of motes that are monitoring an event. In this example a mote is measuring a temperature and relaying this information to a gateway via a multi-hop connection through other motes. The gateway could be a mobile phone that is sending a message via e.g. SMS if it is triggered by the mote indicating that the temperature has risen to or above a certain level.

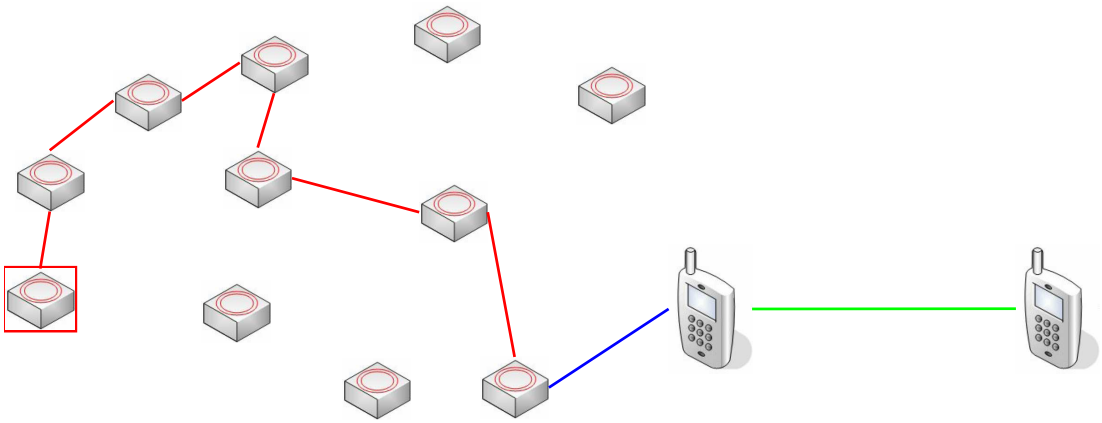


Figure 1.1: *The figure shows a possible setup of motes for an application. A mote (red square) measures the environment and relays the result through the network to a gateway (mobile phone) for viewing or further forwarding.*

Other possible applications for a wireless sensor network are briefly described in the following:

Smoke detection is of great importance in various environments including private homes. Such systems of cooperating smoke detectors already exist. When a possible fire is detected by one sensor it can alert the others setting off the alarm in every one, or the message can be conveyed through the network to a main control station.

Temporary setups of communication or security systems can be a simple solution in situations of urban warfare or natural disasters. Such events often result in broken cellular networks or potential risk for aid workers. Wireless sensor networks is fast and easy to deploy for these purposes

Parking of a car in a small space or driving reverse with accuracy can be improved by attaching ultra sonic distance sensors to the car. By continuous measurements transmitted to a receiver (mobile phone) the driver is able to move the car closer to other cars or objects. This could be a cheap alternative to installing parking cameras in the car.

1.1 First AAU mote

Motes are in theory very cheap but the first prototypes are having a price of approximately 100 US dollars. Aalborg University have developed its first prototype of such a mote which can measure distances by an ultra sonic ranger and broadcast this measurement via Bluetooth. This mote can be seen in Figure 1.2. The goal for AAU is to develop generic motes capable of adapting to a wide set of applications.

Figure 1.2: *The figure is showing the distance sensor mote developed by Aalborg University*

Each mote contains a microprocessor, communication module, sensor module and power supply. All of this is contained in a box typically about the same size as a mobile phone.

The first motes developed at Aalborg university does only support communication via Bluetooth. The Bluetooth module is the most costly part of this mote, therefore it would be ideal also to integrate a low cost ISM band (industrial, medical and scientific) 433 MHz transceiver module. The Bluetooth implementation of the mote does only support a one-hop connection so the idea is to implement the ISM band module so the motes are able to make multi-hop connections as it is seen in Figure 1.1.

1.2 Second AAU mote

A new mote is being developed at Aalborg University and can be seen in Figure 1.3. As seen the mote consists of two different boards stacked on each other which gives the flexibility of adding more features in a later design. This new mote is the one being used in this project and the current one has the features described in the following subsections. See Appendix F for further information about hardware schematics.

Figure 1.3: *The figure is showing the second version of the mote developed by Aalborg University*

1.2.1 Main board

- Li-Poly battery interface which can be used for power supply to the mote
- Mini USB interface which can be used for both serial RS-232 interface to the mote and external power supply. It is also possible to charge the Li-Poly battery by changing the jumper settings on the board.
- dsPIC microprocessor for controlling the mote (further described in Section 1.2.3)
- 22.1 MHz oscillator as external clock source for the dsPIC
- ICD 2 debugger/programmer interface

1.2.2 Wireless board

- Bluetooth module (Amber Wireless AMB2300) for wireless communication with mobile phones or PCs, connected with serial RS-232 connection to the microprocessor
- RF transceiver for communicating via the ISM band. (Further described in Section 1.2.4).
- Loop antenna for the RF transceiver

1.2.3 Microprocessor

The mote is controlled by a microprocessor from Microchip with the product name dsPIC33FJ256GP710 referred to as dsPIC, which is a 16-bit Digital Signal Controller (DSC) based on the modified Harvard architecture. It has the following relevant features that can be utilized in this project:

- 256 KB Flash memory
- 30 KB RAM
- 85 programmable digital I/O pins, 100 pins in total
- Two (Universal Asynchronous Receiver Transmitter) UARTs
- Two (Serial Peripheral Interface) SPIs
- Nine 16-bit timers
- C compiler optimized instruction set, 83 instructions

[3]

One UART is used for RS-232 serial communication and is connected to the USB interface. The second is connected to the Bluetooth module.

1.2.4 RF transceiver

To communicate in the unlicensed ISM band the RF transceiver (Nordic Semiconductor nRF905) referred to as nRF, is connected to SPI1 on the microprocessor. The nRF has the following specifications which are useful in this project:

- Gaussian Frequency Shift Keying (GFSK) modulation, Manchester encoded
- 32 pins
- 8 SPI instructions for configuration
- Maximum transmit output power of 10 dBm (can be varied)
- Transmitted data rate 100 kb/s
- Can be used in the ISM bands 433, 868, or 915 MHz
- Carrier detection mechanism for "listen before talk" protocols
- Data Ready signal when a package is transmitted or received

- Address Match for incoming data detection
- Automatic retransmission
- Automatic Cyclic Redundancy Check (CRC) generation

[8]

The nRF also contains five interval registers which is status, RF configuration, TX address, TX payload and RX payload. The TX address register has a length of four bytes which means that addressing of receivers should be done with one to four byte addresses. The address of transceiver itself is contained in the configuration register also four bytes wide. The payload registers is 32 bytes each which also determines the maximum package size.

1.3 Initial problem

The hardware of the motes is being produced and assembled in parallel with this project by other people at Aalborg University meaning that no hardware will be developed in this project. The current motes does not have any sensing capabilities yet and this project will not aim for a specific measuring application.

As the motes are brand new, no protocols have been developed or implemented for them. Thus the aim of this project is the development and implementation of a Medium Access Control (MAC) protocol for the ISM band module which makes it possible to establish direct and multi-hop connections between the motes.

Chapter 2

Analysis

This chapter concerns issues to be considered when developing and working with wireless networks. When designing a protocol it is essential to be aware of which layers in network communication the protocol deals with. This project is working with the mechanisms of the data link layer which will be described here including classification and examples of different Multiple Access Protocols. The possible problems of wireless networks compared to regular wired networks will be investigated and the proposed mechanisms to solve them will be described. Energy consumption and real time aspects will briefly be discussed as it is also important in wireless sensor networks. This analysis of wireless scenarios will lead to the choice of which type of MAC protocol to design and implement for the second version of the AAU mote.

2.1 The data link layer

The data link layer is the second layer in the OSI reference model for network communication and is often referred to as layer two. It has interfaces to the physical layer and the network layer, 1 and 3 respectively. The layers of the OSI model and the location of the data link layer are shown in Figure 2.1.

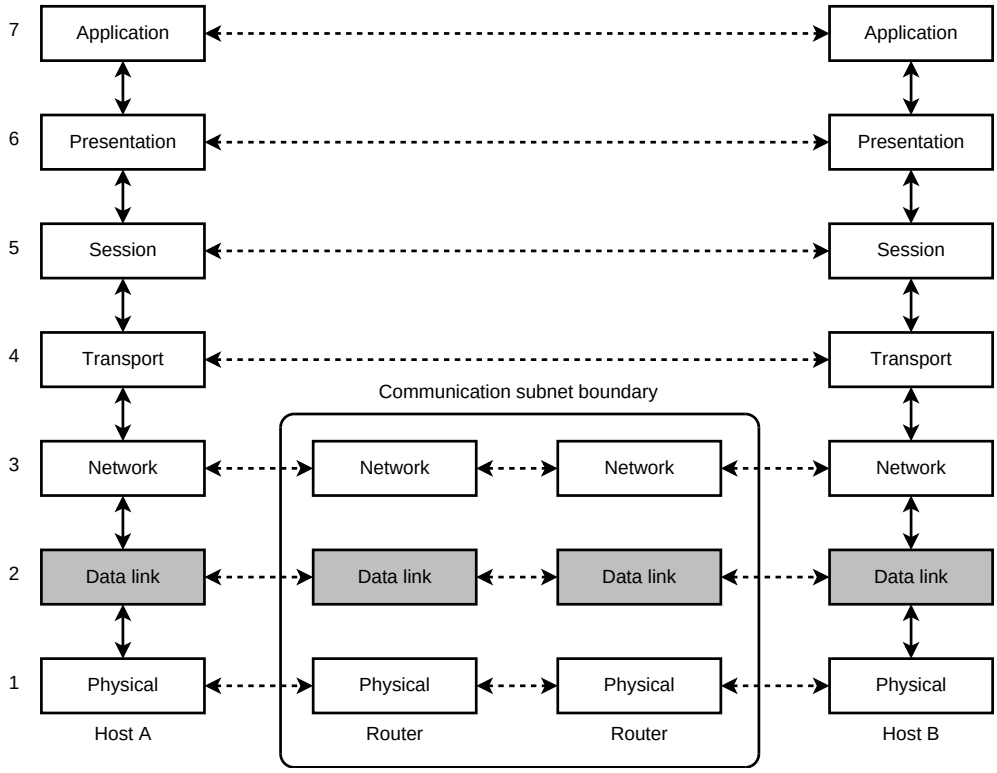


Figure 2.1: *The OSI reference model. The data link layer is layer 2 in the reference model and it deals with node-to-node rather than end-to-end communication [10].*

The job of the data link layer is to provide an error free communication line to the network layer above. This is done at the sender by dividing the raw bit stream into data frames which are sent sequentially to the receiver over a wire-like channel, i.e. a channel that acts like a wire like a cable or a point-to-point wireless link. If the frame is received correctly, the receiver will send an acknowledgement frame back to the sender to inform him about it. The data link layer should also perform flow control of the transmission to prevent slow receivers from getting buffer overflow and thereby loosing data frames. This can be done by using feed back messages from the receiver allowing the sender to continue or slow down the transmission.

As shown in Figure 2.1, data link layer protocols are operating between each machine in a network, i.e. routers or hosts which are interconnected. This is unlike layers 4-7 which features protocols dealing with end-to-end connections making the network and the machines within, transparent.

The MAC layer is a sublayer of the data link layer i.e. it is not represented in the OSI model. This layer is used in networks where multiple machines need to communicate via a single communication channel. The protocols of the layer are called Multiple Access Protocols (MAP) and deals with the task of scheduling and determining which machine or node should have access to the channel next [10].

2.2 Multiple Access Protocols

Starting in 1970 with the Aloha protocol, many algorithms for allocating a multiple access channel have been developed. This section will consider a classification of MAPs and use the Aloha protocol as an example of a simple way to share the used channel. Also carrier sensing is examined as a way of avoiding two nodes transmitting at the same time creating a collision. [5]

2.2.1 Classification of Multiple Access Protocols

At the highest level of the classification there are conflict-free and contention protocols. The classification of MAPs is shown in Figure 2.2. [6]

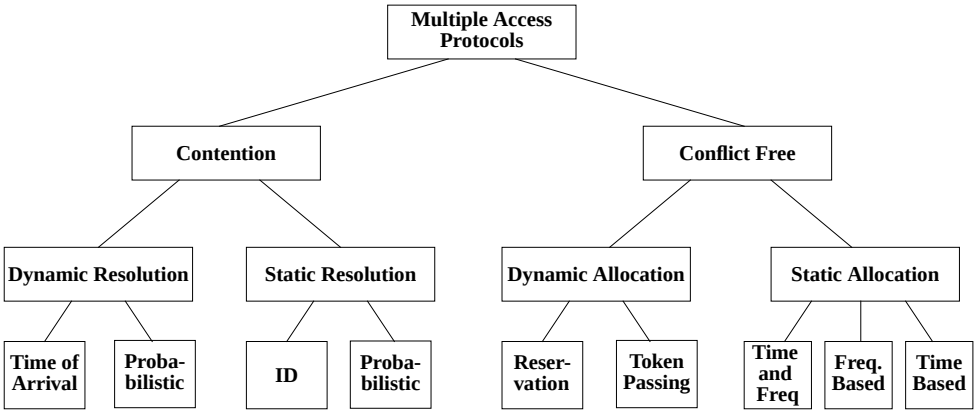


Figure 2.2: *Classification of Multiple Access Protocols [6]*

Conflict free protocols are those scheduling the transmissions of all users [5]. In this way, by adjusting each user's transmitting time or frequency, it avoids that two or more users transmit simultaneously.

Conflict free transmission can be achieved by allocating the channel to the users either statically or dynamically. In the case of the static allocation, whether each user is active or not, the channel capacity is divided among the users and to each user assigned a fix part. Hence the division can be done for a fraction of time like in Time Division Multiple Access (TDMA), where the channel capacity of one slot per frame is assigned to each user. The frequency bands division results in the Frequency Division Multiple Access (FDMA) protocol where a fixed band is assigned to each user. The principles of FDMA and TDMA are shown in Figure 2.3 [6].

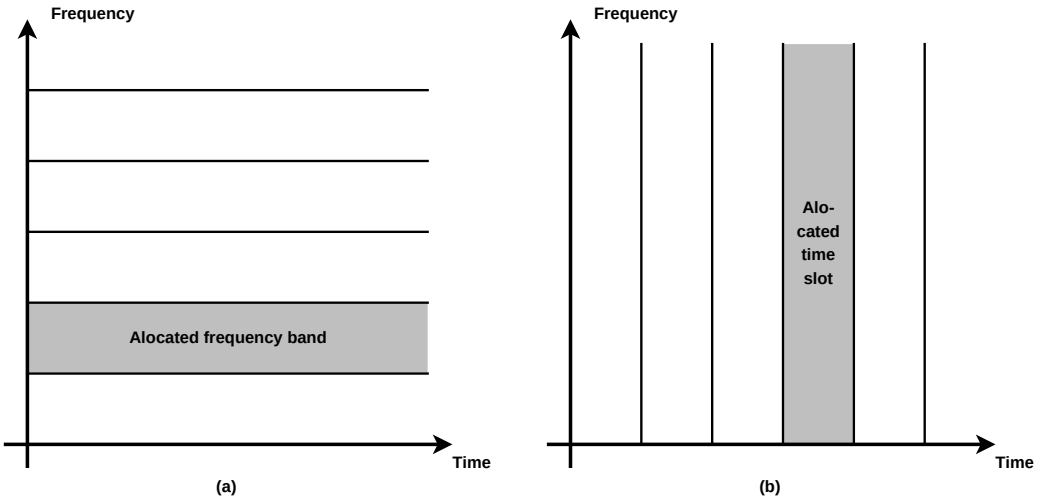


Figure 2.3: *(a) The division of bandwidth in FDMA and (b) the division of time in TDMA*

The dynamic allocation assign a channel only to a user who has something to transmit. Thus, the user without transmitting data does not waste the channel capacity. This allocation can be further classified, based on the assignment scheduling, into reservation and token passing schemes. With reservation schemes, the users first announce their intent to transmit and all those who have so announced will transmit before new users have a chance to announce their intent to transmit. With token passing schemes, a special frame (Token) is passed in order from one terminal to another terminal permitting only the token holder to transmit [6].

Contention protocol schemes differ from conflict free schemes since there is no scheduling of transmissions. Hence, collisions may occur and the protocol should be able to solve those conflicts when several users transmit simultaneously. Also the resolution process together with the idle users consume channel resources, which is a major difference between various contention protocols.

In order to guarantee a successful transmission, it is necessary to find a way to avoid collisions. Also here a distinction between static and dynamic resolutions can be made. Static resolution means that the dynamics of the network does not have any influence on the behavior of the system. The static resolution can be either probabilistic, meaning that the transmission of a packet happens with a fixed probability, or based on ID, meaning that users have different priority in the network. The dynamic resolution can prioritize packets based on time of arrival or be probabilistic, but with a dynamic probability changing as a result of the interference in the network [6].

Both classes have advantages and disadvantages regarding resource usage, throughput and scalability. Some of these are listed in Table 2.1.

MAP class	**A**dvantages	**D**isadvantages
Conflict free	No transmission interference	Low throughput for each user
	Fair division of capacity	Unused resources for idle users
Contention	Efficient for "bursty" users	Resource consumption for error correction
	Efficient in ad-hoc networks	Possible delay and unfair capacity division

Table 2.1: *Some of the advantages and disadvantages for the two classes of MAPs*

2.2.2 Aloha

The Aloha protocol was used on Hawaii in the early 1970ties and was one of the first design of a computer network via a shared medium (radio). The system was build on a hub/star typology and used two different frequencies where the hub broadcasted on the first one and the clients were transmitting on the other frequency. The basic idea of this protocol is:

- If a client has a packet to send, it will transmit it.
- In case of collision in this transmission, the client will try to resend the packet later.

This means that if the packet is successfully received by the hub, it immediately replies with the same packet as an acknowledgement. If the client never receives this reply the result is a collision and a retransmission must be made. The principle is called Pure Aloha and can be seen in Figure 2.4. Pure Aloha does only have a maximum throughput of about 18.4% due to collisions.

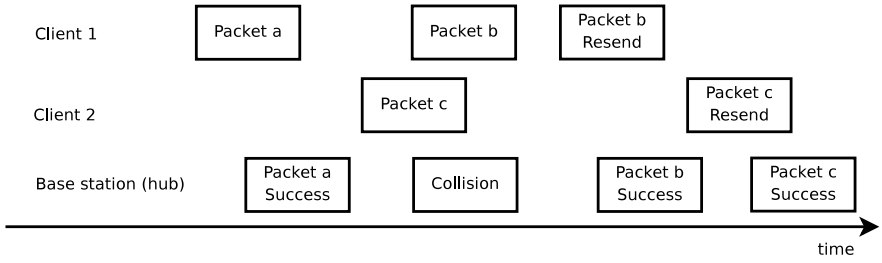


Figure 2.4: *An example of Pure Aloha with 2 client and a base station [4]*

Later this throughput was doubled to 36.8% by introducing the principle of Slotted Aloha. In slotted version of Aloha timeslots are introduced where a centralized clock transmits a tick in the beginning of each slot. The clients can only transmit when a tick is received (beginning of a new slot), this can be seen in Figure 2.5 [13].

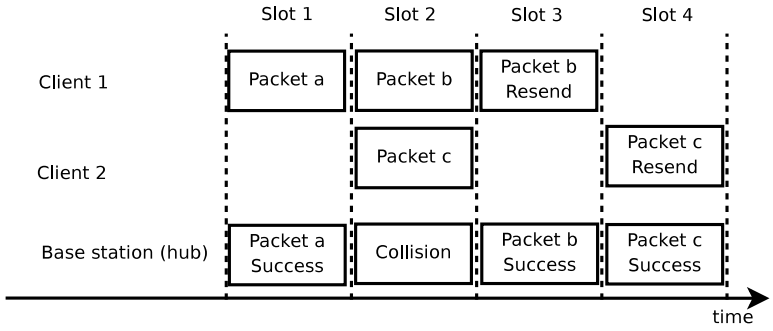


Figure 2.5: *An example of Slotted Aloha with 2 client and a base station [4]*

2.2.3 CSMA protocols

An improvement to the Pure Aloha is to sense the carrier before accessing the medium. Protocols in which a node verifies the absence of other traffic before transmitting are called Carrier Sense Multiple Access (CSMA).

Carrier Sense describes the fact that, before a node transmits, it "listens" to the medium to determine if another node in the neighborhood is transmitting on the same channel. If the medium is quiet, the node recognizes that this is an appropriate time to transmit. If a carrier is sensed, the node waits for the transmission in progress to finish before initiating its own transmission. In this way, the probability of a collision decreases.[7]

Multiple Access describes the fact that multiple nodes send and receive on the medium. Transmissions by one node are generally received by all other nodes using the same medium. There are different variations of the CSMA protocol which is described below.

1-persistent CSMA
When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment. If the channel is idle, the node transmits a packet immediately with a probability of 1. If the channel is busy, the node keeps listening and transmit immediately when the channel becomes idle. As soon as the channel becomes idle, all the nodes wishing to transmit access the medium at the same time. Collisions can occur only when more than one user begins transmitting within the period of propagation delay. Even if the propagation delay is zero, there will still be collisions because of the time from sensing the idle carrier to the transmission starts [10] [7].

Non-persistent CSMA
To send data, a node first listens to the channel to see if anyone else is transmitting and starts sending immediately if the medium is idle. If the medium is busy, the node waits a random amount of time and sense the channel again. Consequently, this algorithm leads to better channel utilization but longer delays than 1-persistent CSMA [10].

P-persistent CSMA

In p-persistent CSMA, the nodes also sense the medium before sending. If the channel is idle, transmit a packet with probability p and delay for one time slot with probability (1-p) and start over. If the channel is busy, then delay one time-slot and start over. Figure 2.6 shows the different states in p-persistent CSMA. [10]

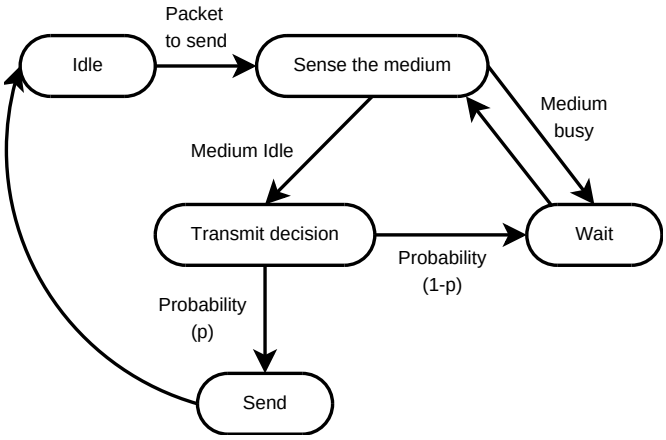


Figure 2.6: *State diagram showing the principles of p-persistent CSMA.*

CSMA and Aloha comparison

Figure 2.7 shows the computed throughput versus offered traffic for all three x-persistent CSMA protocols, as well as for pure and slotted Aloha. In this figure the throughput S on the y-axis represents the expected number of successful transmissions per packet. The load G in the x-axis represents the number of attempted transmissions. Due to the possibility of collisions the load is usually bigger than the throughput. For example the throughput S for pure Aloha is $S = Ge^{-2G}$ and as seen on the figure it has a maximum value of $S = 1/2e = 0.184$ when the load is equal to 0.5. The slotted Aloha instead has a throughput of $S = Ge^{-G}$. When the load is equal to 1, S has its maximum value of $S = 0.368$ that is the one of pure Aloha. Figure 2.7 also shows how the CSMA protocols have a better throughput than Aloha protocols. [10]

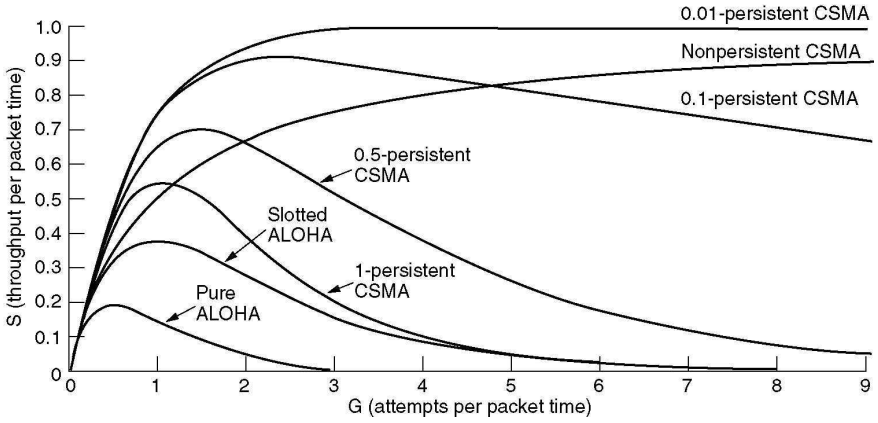


Figure 2.7: *Comparison of the channel utilization versus load for various random access protocols [10].*

CSMA with Collision Detection

In Carrier Sense Multiple Access With Collision Detection (CSMA/CD), if a collision occurs, the first node which detects the collision sends a jam signal to all stations to indicate that there has been a collision. After receiving a jam signal, a node that was attempting to transmit abort its transmission and tries again later after waiting a random amount of time. The minimum time to detect the collision is the time it takes the signal to propagate from one station to the other and the maximum time needed is two times the propagation delay. This results in a much more efficient use of the media since the bandwidth of transmitting the entire frame is not wasted [7].

2.3 Multiple access in wireless networks

CSMA/CD scheme is a widely used MAC scheme for wired networks, but the use of this protocol in wireless networks results in additional problems. CSMA/CD is not really interested in collisions at the sender, but rather in those at the receiver. The signal should reach the receiver without collisions. But the sender is the one detecting collisions. The difference here is in the signal strength, which remains almost the same for wired networks. For wireless networks, the signal strength decreases proportionally to the square of the distance to the transmitter. Obstacles in the line of sight attenuate the signal even further. This means that the collision at the receiver due to another sender, in many cases goes undetected at the sender. As the transmission power in the area of the transmitting antenna is much higher than the receiving power, collision detection is very difficult in wireless scenarios, and in practice not possible. There are several other issues to consider when moving from the wired domain into the wireless. Some of these are described in the following [7].

2.3.1 Hidden and Exposed terminal problems

Figure 2.8 illustrates the hidden terminal problem. B is in the transmission range of A and C but C is not in transmission range of A and A is not in transmission range of C. Suppose that nodes A and C both want to transmit data to node B. They will both sense the medium free and transmit causing a collision at B. Hence, A is a hidden terminal for C and vice versa [7].

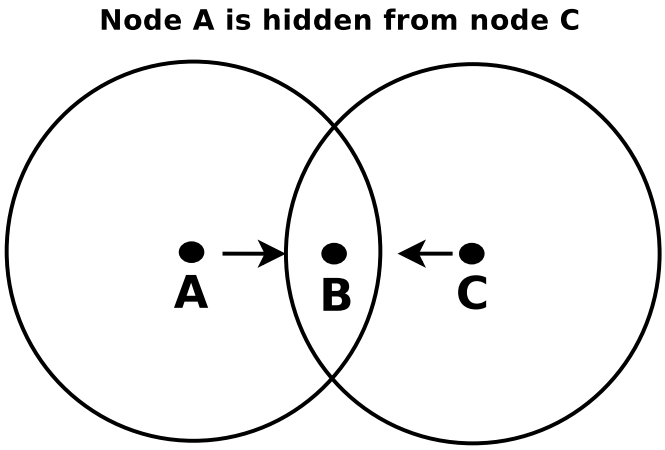


Figure 2.8: *Node A and C are not in transmission range of each other. Thus they are hidden terminals to each other.*

Figure 2.9 illustrates the exposed terminal problem. B is in transmission range of A and C, and C is in transmission range of B and D. Suppose that node B is sending a packet to node A and C intends to transmit data to node D. C senses the medium to be busy and will not send any packet, postponing its transmission. In reality, no collision would have happened at A because A is outside the transmission range of C. Hence, this problem causes unnecessary delay. This means that C is exposed to B [7].

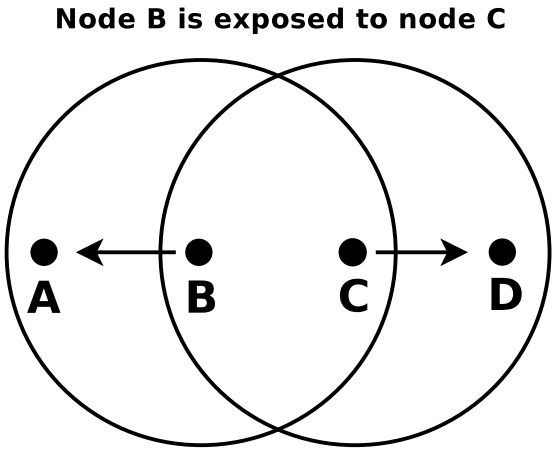


Figure 2.9: *Node B and C is in transmission range of each other and want to send in different directions. Thus they are exposed terminals to each other.*

2.3.2 Near and far terminals

Figure 2.10 illustrates the near-far terminal problem. Suppose that A and B are both sending with the same transmission power. As the signal strength decreases proportionally to the square of the distance, B's signal drowns out A's signal. As a result, C cannot receive A's transmission. Thus, precise power control is needed to receive all senders with the same strength at the receiver [7].

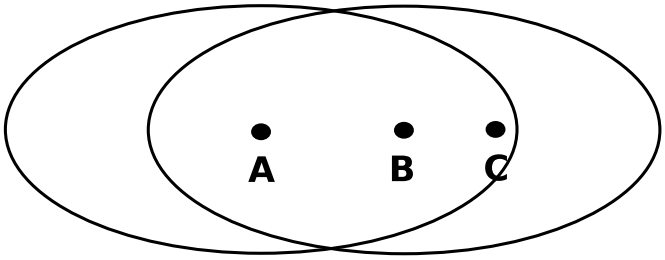


Figure 2.10: *Near and far terminals. Node A can not send to node C if B is transmitting as B's signal will drown out the one from A.*

2.3.3 Multiple Access with Collision Avoidance

CSMA with Collision Avoidance (CSMA/CA) is used to improve the performance of pure CSMA. Like in CSMA a node wishing to transmit will first listen to the channel for a predetermined amount of time. If the channel is sensed idle, the node will start its transmission. CSMA/CA is used where CSMA/CD cannot be implemented due to the nature of the channel. CSMA/CA implements a Request To Send / Clear To Send (RTS/CTS) mechanism known as the IEEE 802.11 RTS/CTS exchange.

Multiple Access with Collision Avoidance (MACA) is a MAC protocol used in wireless LAN data transmission to avoid collisions caused by the hidden terminal problem and to simplify exposed terminal problem. It is inspired by the mechanisms of CSMA/CA, but does not implement carrier sensing (which leaves the name MA/CA or simply MACA) [2].

In this scheme, the node that needs to transmit a message sends a small RTS message to the receiver. The receiver immediately replies with a small CTS message to the sender. After receiving the CTS, the sender will transmit the data message. Both the RTS and the CTS messages carry the length of (or time to transmit) the data message as well as the names of sender and receiver [10].

Meanwhile, any node hearing the RTS must remain silent during the time needed for the other nodes to exchange CTS message and data packet. Any node hearing the CTS must remain silent until the data transmission is complete. Figure 2.11 shows how MACA can solve the hidden terminal problem.[7]

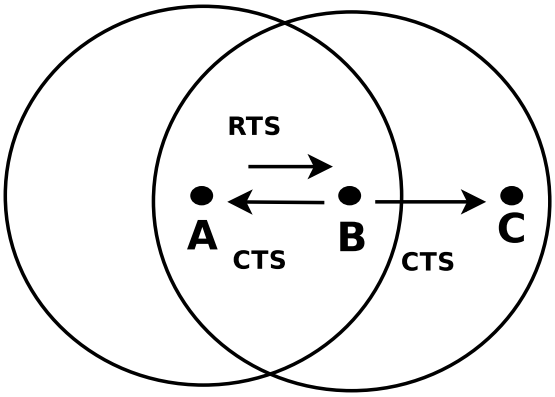


Figure 2.11: *The RTS/CTS exchange can solve the hidden terminal problem. The figure shows an exchange between node A and B with the CTS overheard by node C*

A wants to send to B and C is only in the transmission range of B. A send RTS which is only heard by B, and B responds with CTS heard by both A and C. Thus C will defer its transmission for the duration indicated in the CTS toward B. Still, collisions can occur during the sending of an RTS (i.e. A and C might send RTS simultaneously). But RTS packets are very small compared to the data transmission, and therefore unlikely to collide.

MACA can also help solving the exposed terminal problem as shown in Figure 2.12.

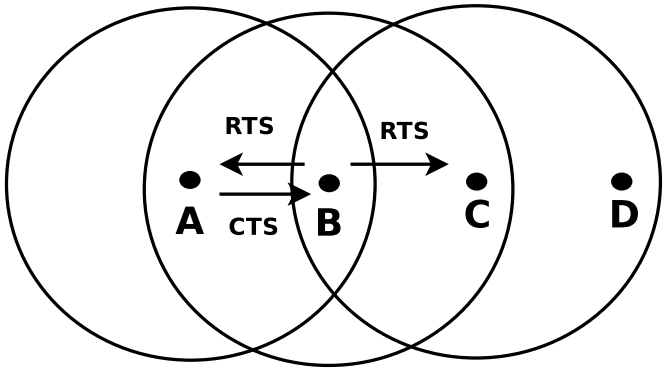


Figure 2.12: *The RTS/CTS exchange can solve the exposed terminal problem. C is allowed to send to D is only RTS, but not CTS is heard.*

In the case of B wanting to send to A and C wanting to send to D, C will hear an RTS message from B to A containing the name of the receiver A and the sender B. Thus C will defer its transmission to D, but if C is not hearing a the corresponding CTS message it will not be in range of the receiver A. I.e. C only waits for the time of a CTS to arrive and is allowed to start a transmission to D if no CTS arrives.

Figure 2.13 shows a simplified state diagram for a sender and receiver that could realize MACA.[7]

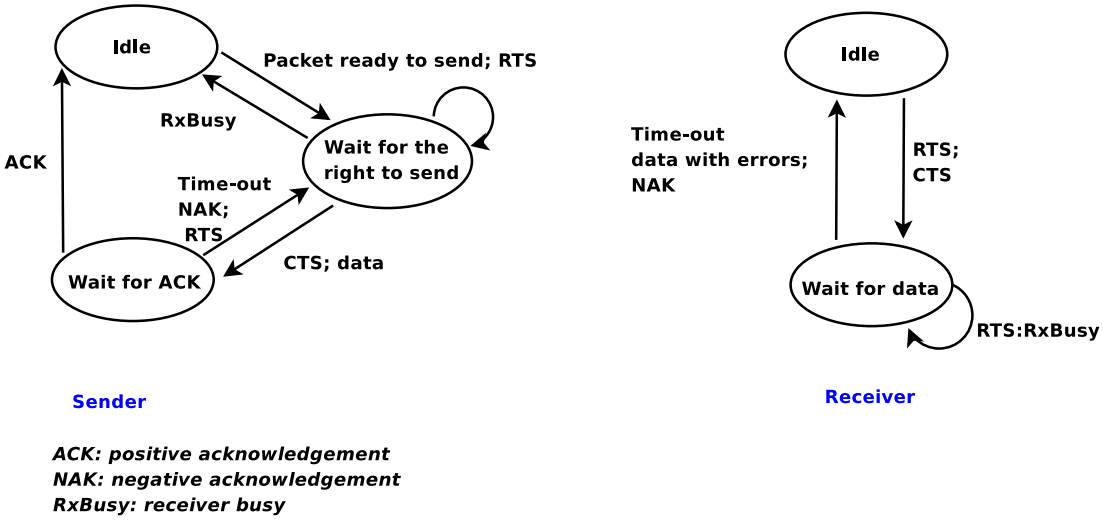


Figure 2.13: *State diagrams for MACA with included acknowledgement [7]*

MACA also introduces a bypass of the RTS/CTS dialog. RTS and CTS packet should be very small compared to the data packet for optimal performance and minimal overheard. If the MAC protocol implementing MACA allows for varying data packet sizes, small data packets about the same size as the RTS and CTS packets could occur. In this case the RTS/CTS exchange

may be bypassed to send the data packet faster, i.e. the data packet is send immediately to the receiver. The sender must of course wait if RTS or CTS packets from other nodes is received.

2.3.4 MACAW

MACAW is a slotted MAC protocol highly used in ad-hoc networks. Furthermore many Wireless sensor network's MAC protocols are based on it, i.e. S-MAC [12]. In the protocol, collision avoidance (RTS/CTS) is used, together with acknowledgement (ACK) to provide solution to the hidden terminal problem. However, MACAW does not adopt carrier sensing, but is using a different approach: Just before sending a data packet, the node sends a short Data-Sending packet (DS) to let nearby nodes know that the RTS/CTS exchange was successful [11].

The MACAW protocol was introduced to extend the function of the MACA protocol. Figure 2.14 illustrates an example of the MACAW protocol. It is assumed that only adjacent nodes are in transmission range of each other.

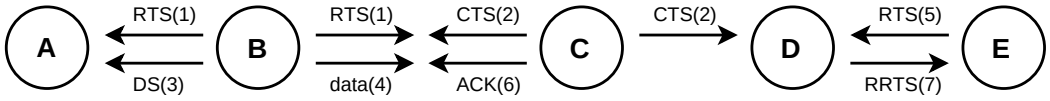


Figure 2.14: *Principles of MACAW. Only adjacent nodes are in transmission range of each other.*

Assume that node B has data to transfer to node C. A successful data transfer in a network containing 6 nodes consists of the following sequence of frames:

1. B sends RTS to C which is also heard by A
2. C replies with CTS which is also heard by D
3. B will now send DS frame to inform A that the RTS/CTS exchange was successful
4. B starts sending the data packet to C
5. Suppose that E now has data to send to D and sends RTS. D can hear the RTS but not reply due to the ongoing transmission to C
6. After successful transmission, C sends ACK to B
7. D is now allowed to transmit and sends a Request for Request To Send (RRTS) to inform E about the idle channel

MACAW is a non-persistent MAC protocol meaning that if the message to transmit contains more than one packet, the node A has to wait a random time after each successful data transfer and then compete with the adjacent nodes again for the medium using the RTS/CTS mechanism.

2.4 Other sensor network issues

Sensor networks may have different requirements depending on which application they should be used for. Requirements may also depend on the surrounding environment and the sensor hardware. This section addresses other issues to consider when designing a MAC protocol for sensor networks.

2.4.1 Energy consumption

Energy efficiency is an important attribute for sensor network MAC protocols, with large numbers of battery powered nodes, it is very difficult to change or recharge batteries for these nodes, so prolonging the lifetime of each node is a critical issue. Energy consumption occurs in three domains: sensing, data processing, and communications. The major sources of energy waste are:

- Collisions
- Idle listening
- Overhearing - When a nodes get a packet for another node
- Protocol overhead - Control frames do not carry useful information although their transmission consumes energy
- Adaptation - Reconfiguring when nodes join and leave the network

It is possible to use the network density and the consequent redundant information to save energy. Some sensors can be switched off while others which cover about all the affected area can be kept awake. It is necessary to employ a technique that manages the sleeping-listening period of the nodes. An effective method to implement this can be grouping nodes together in clusters so that each of them covers an increased area of the territory. These clusters can be put in sleeping/listening mode according to predetermined settings. During sleep, a node turns off its radio, and sets a timer to awake itself later.

The scheme is shown in Figure 2.15. Each node goes to sleep for some time, and then wakes up and listens to see if any other node wants to talk to it. [12]

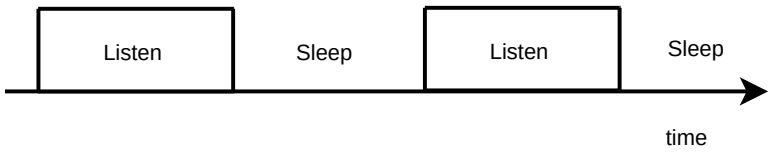


Figure 2.15: *Periodic listen and sleep transition to save energy [12]*

2.4.2 Real time

Some applications using sensor networks may have timing requirements for optimal performance. Examples of this could be applications with continuous flow of data through the network. E.g. voice traffic or measurements of a rapid changing process. In these applications it is essential that the most recent data is available, but loss of individual packets is not important. This effects the design of a MAC protocol i.e. such a protocol should not handle retransmission of packets.

The protocol should be optimized for minimum delay rather than robustness. Medium reservation with RTS/CTS packets should be avoided as they introduce an unnecessary overhead causing delay. Acknowledgement packets is also not needed as no retransmissions should occur. Thus a 1-persistant CSMA protocol would be the best solution for a real time sensor network.

2.5 Problem statement

After investigating these issues regarding development of wireless networks, it will be decided which type of MAC protocol to continue designing. Additionally it will be decided which subjects not to be considered any further. This project aims at developing a MAC protocol and also to test an implementation of this, but at this point it is not the intention to make the protocol suited for direct deployment in a final system. Because of that, the implementation will be tested and demonstrated with a stationary power supply to each mote in the system and the issue of energy consumption will not be considered in the design.

It is desired to make a simple and robust communication protocol with minimal retransmissions and loss of data, which can be achieved by avoiding collisions. Throughput and latency is of minor importance in the first design iteration which will not include real time aspects. Thus it is decided to work on designing and implementing a collision avoidance scheme (RTS/CTS) with acknowledgements. Carrier sensing will also be added because the ISM band is used by many other devices and networks which can cause collisions. Also collisions of RTS packets can be minimized in this way.

To test the network and protocol performance, and to visualize the potential use of a sensor network as a communication network, a mobile phone application utilizing the network will also be developed. The complete system can be used to extend the range of phone-to-phone communication via Bluetooth.

Chapter 3

Requirements specification

It is essential to establish correct requirements and specifications early in the development process to prevent errors later on in the system life cycle. Therefore, the first step in the design process is to establish the preconditions of the system in order to generate the requirements specifications regarding the MAC protocol and the mobile phone application. It is also important to know how to verify that the requirements are correctly implemented. The acceptance test specification in Appendix A describes how this can be done.

3.1 Preconditions

The MAC protocol will be designed for the second mote developed by Aalborg University which means that hardware requirements are already given and implemented as a base of this project. I.e. some protocol requirements are created to fit the hardware of the motes regarding e.g. packet size and transmission range. The requirements of the system also aims at creating a test scenario for both the protocol and the complete system, i.e. a scenario where routing is fixed and motes do not move in space from their starting point. For communication between motes and mobile phone it is required for the mobile phone to support Bluetooth serial connection.

3.2 System requirements

1. The system must consist of a network of at least 4 motes and 2 mobile phones.
2. The two motes at the border of the network must have Bluetooth interfaces to communicate with the mobile phones.
3. The system must be able to convey either text, image or audio files from one mobile phone through the multihop network of motes to the other mobile phone.

3.3 Mote and protocol requirements

1. The protocol must implement 1-persistent carrier sensing.
2. The protocol must implement collision avoidance (RTS/CTS) scheme.
3. The protocol must implement packet acknowledgements (ACK).
4. If timeout on ACK packets occur, the RTS/CTS exchange must be done again.
5. Packet payload size must be maximum 29 bytes.
6. The first three bytes in each packet must be used as header.
7. Each motes's ISM interface must be identified by a 32 bit address.
8. For testing purposes each mote must have a transmission range of maximum 50 cm.
9. The motes must be powered by a static power supply or batteries.
10. The motes with Bluetooth module must be able to accept a connection from a mobile phone.
11. The network must convey all non-real-time traffic, meaning no specific time of arrival will be guaranteed.
12. All data must arrive correctly at the receiver, i.e. duplicated packets must not occur.

3.4 Mobile phone application requirements

1. At least one mote with Bluetooth interface must be less far than 10 metres from the mobile phone.
2. A sender application and a receiver application must be created.
3. A GUI must be created for the sender application.
4. A GUI must be created for the receiver application.
5. The sender function must be able to:
   (a) Create text files.
   (b) Create image files.
   (c) Create audio files.
   (d) Search for Bluetooth devices.
   (e) Display the devices to which it is possible to establish a connection.

   (f) Permit to the user to select the desired device.

   (g) Establish a connection to the selected device.

   (h) Send the messages.

6. The receiver function must be able to:

   (a) Search for Bluetooth devices.

   (b) Display the devices to which it is possible to establish a connection.

   (c) Permit to the user to select the desired device.

   (d) Establish a connection to the selected device.

   (e) Listen for incoming messages until the program is closed.

   (f) Show the received message.

7. The data to send must be split into packets with a size of 29 bytes.

Chapter 4

System design

In this chapter the design of the whole system including the motes and the two mobile phones will be described to satisfy the specified requirements. First a general scenario of the network is outlined to show how the system can be used as a real application. Then the interfaces in the network are defined as well as the communication flow between the individual devices in the system.

4.1 Network scenario

The goal of this project is to design a MAC protocol that can provide a robust connection between two mobile phones, via a wireless mote line network topology as shown in Figure 4.1. When the mobile phone needs to transmit something, the message will be directed to the first mote in the network through a Bluetooth connection. The motes will communicate over the ISM interface to transmit a unidirectional flow of messages through the multi hop network. The second mobile phone will connect to the fixed wireless mote network via its Bluetooth interface to receive the message.

The motes have different roles in the network: The first mote is an input mote (I), the motes transmitting the message through the network are called forwarders (F), while the last mote is an output mote (O). Depending on the role, the mote knows where to send the packets.

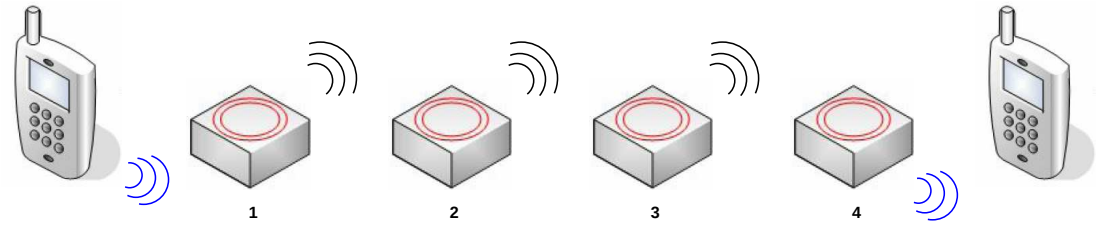


Figure 4.1: *The network scenario is a line network topology with 2 mobile phones and 4 motes.*

The routing is static, with the advantage of being predictable, and simple to set up. It will be easy to manage in the small network but does not scale well in large networks and does not dynamically adapt to network topology changes or equipment failures. Ad-hoc communication between random nodes in the network will be excluded. In the test scenario in Figure 4.1 it is also assumed that the nodes are not moving and at a random point in time during transmission, all motes have packets they want to send.

The mote network consist of 4 motes as shown in Figure 4.1. The number of motes is chosen to make the smallest network possible because only a limited number of motes is available. To test the avoidance of the hidden terminal problem at least 4 motes in the network is needed. It will not be possible to test the avoidance of the exposed terminal problem because the flow of data is unidirectional.

Messages flow from the first node to the last one in a line topology, so node 1 sends only to 2, 2 only to 3, and 3 only to 4 and so on. The communication in the network follows the steps below:

1. Initialization of mobile phone to mote and mote to mobile phone connection via the Bluetooth interface.

2. The mobile phone will send the data only after receiving a request from mote 1 which indicates that its buffer is not full. If this is the case, new packets may be sent to the mote.

3. When there are no more packets to transmit, mote 1 will wait for timeout, and close the Bluetooth connection to the phone.

4. Before forwarding the packet in the network, the mote broadcasts an RTS packet to its local neighborhood containing the sender and receiver.

5. When mote 2 hears an RTS packet for data destined for it from node 1, it broadcasts a CTS packet to its local neighborhood, similar in structure to the RTS.

6. Node 1 sends a data packet to node 2.

7. Node 2 responds with an ACK packet.

8. This pattern continues throughout the network until the last mote (4) is reached.

9. Mote 4 will send the packets to the receiving mobile phone and wait for acknowledgement for each packet. This can be done because the switching between send and receive for Bluetooth is very fast (approximately 20 ms). See Appendix C.

10. The mobile phone will close the connection when all the packets are transmitted and inform the last mote about this by an end message.

Before the transmission of each packet, a carrier sensing (CS) of the medium is needed. If the channel is idle, the nodes will transmit the packet with probability of 1. If the channel is busy, the nodes will keep sensing the medium and begin transmission as soon as the channel becomes idle. Figure 4.2 shows the flow of a packet in the network.

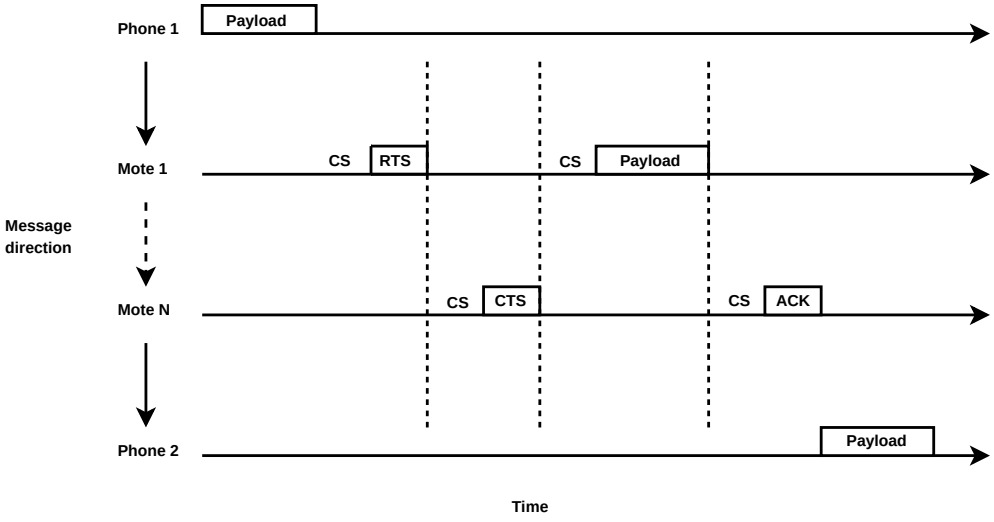


Figure 4.2: *The flow of one payload packet through a network of N motes.*

Even though collisions and loss of packets should not occur in the network it can happen that control packets (RTS, CTS and ACK) are lost. In the case of RTS and CTS the result would just be a retransmission of RTS and a small delay, but if ACK is lost the payload will be retransmitted and duplicated at the receiver mote. To prevent this a sequence number can be introduced. Figure 4.3 illustrates this problem and the solution with the sequence number.

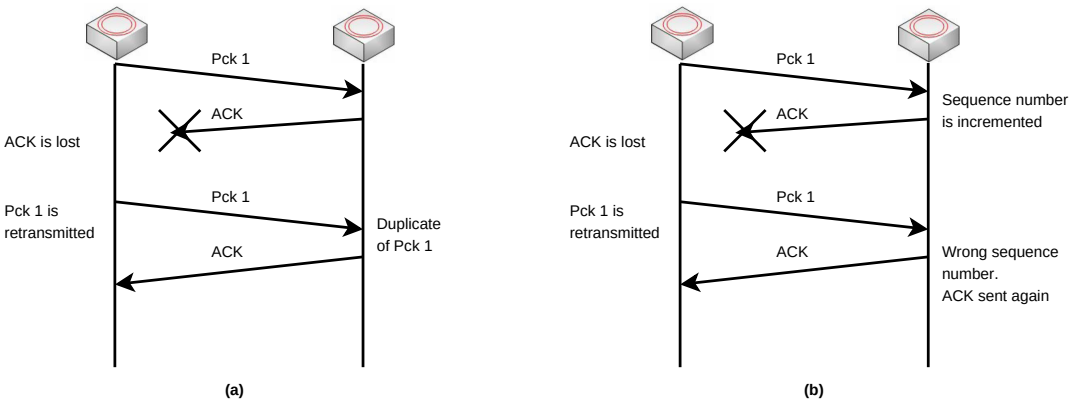


Figure 4.3: *(a) When ACK is lost, the packet is retransmitted and duplicated on the receiver mote. (b) When introducing sequence numbers, the duplicate packet will be discarted on the receiver mote because the sequence number is different from the expected one. The lost ACK will be sent again.*

The network is designed to be FIFO (First In, First Out) i.e. the packets leave the network in the same order as they enter. This means that only a single bit is needed for the sequence number which will make each packet different from the one before and after.

On each dsPIC microprocessor the same software should be implemented. However, each mote must be configured individually, specifying its role in the network: I, O or F, as well as its neighbors.

4.2 Network interfaces

In this section the design of the mote header is described regarding the mechanisms of RTS/CTS/PAYLOAD/ACK packets. Also the network interfaces between the mobile phones and the motes in the network will be explained.

4.2.1 Header design

The actual address of the packets sent between two motes is determined by the nRF and can be seen to the right in Figure B.1 of Appendix B. A Data Package contains a destination address plus the payload. Preamble and CRC are generated and checked automatically and will not be considered further.

The flow chart in Figure B.1 shows that a packet will only be received if the the address matches the mote. Otherwise the nRF will continue listening for a correct address. This means that if all motes must be able to detect and receive all transmissions, the nRF of the motes must all have the same address. The unique mote address must be set and stored in the memory of the dsPIC and the header of each packet will be the first three bytes of the nRF payload.

The header will be designed to inform about sending and receiving mote as well as different flags indicating packet options. The length of each mote's address is one byte, thus the first two bytes are assigned for sender and receiver address. This means that the protocol can support a network of 256 motes. The last byte in the header is used for the options: type, sequence number and buffer status.

The packet type is defined by the following four bits, where a bit value of one means enable:

- Bit 1 - RTS
- Bit 2 - CTS
- Bit 3 - ACK
- Bit 4 - PAYLOAD

As explained in Section 4.1 one bit is used to define the sequence number where packet n has sequence number $nmod2$. One bit is used to define buffer status, where 1 means full and 0 means not full. The two last bits are reserved for future use.

The remaining 29 bytes in the packets are used for payload. A full payload packet and the specified header can be seen in Figure 4.4. The header is added when the first mote receives the payload from the mobile phone and removed when the last mote forwards the packet to the receiver mobile phone.

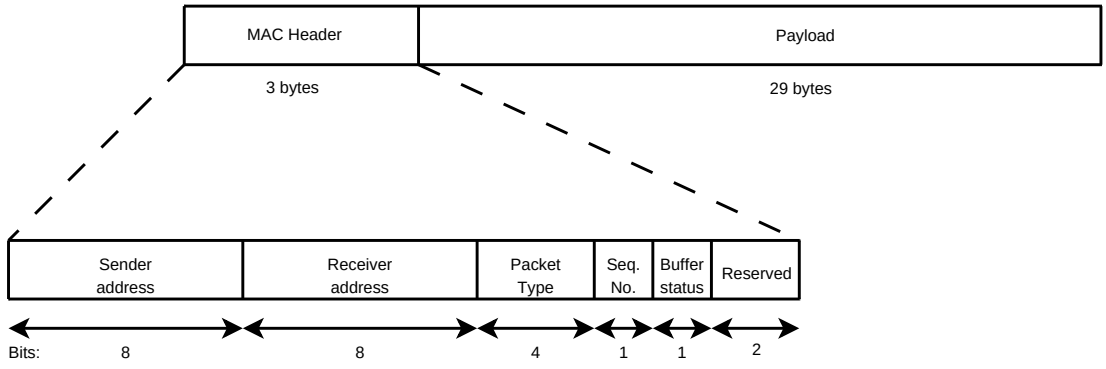


Figure 4.4: *The first figure is showing a full payload packet, where three bytes are assigned as header. The second shows the header where the different bits are assigned*

The transmission time is not built into the RTS and CTS packets because every payload is 29 bytes and the transmission time should be the same for every packet.

4.2.2 Mobile to mote

When the mobile phone has something to send it will initialize a Bluetooth connection to the input mote by sending an initialization message (INIT\n). Afterwards the mobile phone will

wait on a request (REQ) from the mote. After receiving this request which means that there is space in the buffer the mobile phone will send a packet (Pck) and wait for a new request. When all packets are sent and if a timeout occurs, the mote will end the connection. A time line of these events can be seen in Figure 4.5.

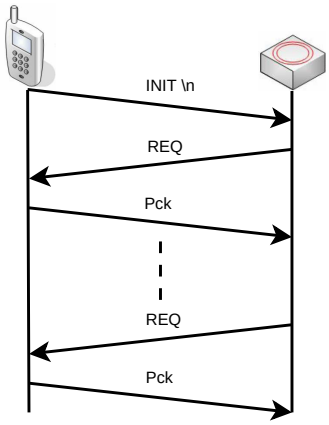


Figure 4.5: *Communication flow between the sender mobile phone and the input mote*

The commands used in the interface between mobile phone to mote are the following text strings:

- `INIT\n` - Initialization for sending
- `REQ` - This command is sent from the mote when there is empty space in the buffer.

When a packet is received the mote will add the header with sender address, receiver address, payload flag and sequence number, whereafter it is put in the buffer.

4.2.3 Mote to mote

When a mote receives a payload packet from another mote, it changes the addresses in the header and sends a packet with the RTS flag enabled to the next mote. The receiving mote will reply by sending a packet with CTS enabled and buffer full enabled/disabled. If the CTS packet has buffer full disabled the sending mote will forward the packet with payload, otherwise the sending mote needs to send an RTS packet again. When the receiver mote have received the packet with payload it will send an ACK packet including buffer status. This can be seen in Figure 4.6.

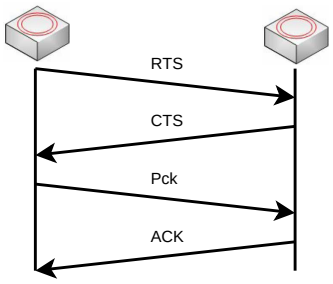


Figure 4.6: *Communication flow between two motes in the network*

The header and header flags are set in the following way:

- RTS packet: Sender address, receiver address, RTS enable
- CTS packet: Sender address, receiver address, CTS enable and buffer status enabled/disabled
- ACK packet: Sender address, receiver address, ACK enable and buffer status enabled/disabled
- Payload packet: Sender address, receiver address, PAYLOAD enable, sequence number and 29 bytes of payload

4.2.4 Mote to mobile

The connection between the last mote and the receiver mobile phone is assumed to be established prior to the start of the packet flow. When the mote have a packet to send it will first wait for a connect message (INIT\n) and then transmit the data to the mobile phone which will send an ACK back. If the acknowledgement is not received the packet will be retransmitted. When the mobile phone have received all packets it will terminate the connection by sending an END message instead of ACK. The flow between the last mote and the receiving mobile phone can be seen in Figure 4.7.

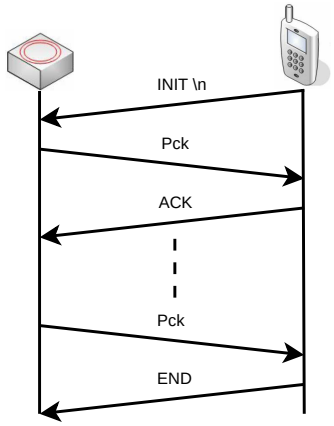


Figure 4.7: *Communication flow between the end gateway mote and the receiver mobile phone*

The commands used in the interface between mobile phone to mote are the following text strings:

- `INIT\n` - Initialization for receiving
- `ACK` - Acknowledgement
- `END` - Termination of the connection

Chapter 5

Mote protocol design

In this chapter the design of the program to be implemented on the dsPIC will be described using activity diagrams. These has been made to outline the different sequences on the receiver and sender side, as well as the main program. This chapter will also contain specific information about the initialization and the functions of the different modules in the program. A module in the program is defined as a collection of functionality to accomplish a task. Each module will described with respect to initialization and functions of the module.

5.1 Program design

The design of the program is documented through several activity diagrams which will be described in this section. Several acronyms used in this section will be described below. These are output signals from the nRF to be read by the dsPIC.

AM: Address Match - Indicates an incoming packet with correct ISM interface address

DR: Data Ready - Indicates that a correct packet is received and ready to be read by the dsPIC

CD: Carrier Detect - Indicates that the carrier frequency is busy

5.1.1 Main flow

Before the motes are ready to send or receive data packets, the various subsystems of the mote must be initialized. I.e. the two UARTs, the SPI and the pin I/O configuration of the dsPIC plus the configuration of the nRF. The motes are not capable of transmitting and receiving simultaneously, but a mote must try to forward the packets in the buffer and listen for new

incoming packets concurrently. Thus the mote must monitor the channel before it tries to send a packet to minimize the possibility of pending incoming packets. If incoming packets are detected, they are expected to be RTS/CTS according to the protocol and the mote will start receiving if the address is correct or backoff otherwise. If the mote has packets in the buffer and the channel seems to be idle, it will start transmitting. These actions is shown in Figure 5.1 and a description of the flow is given below.

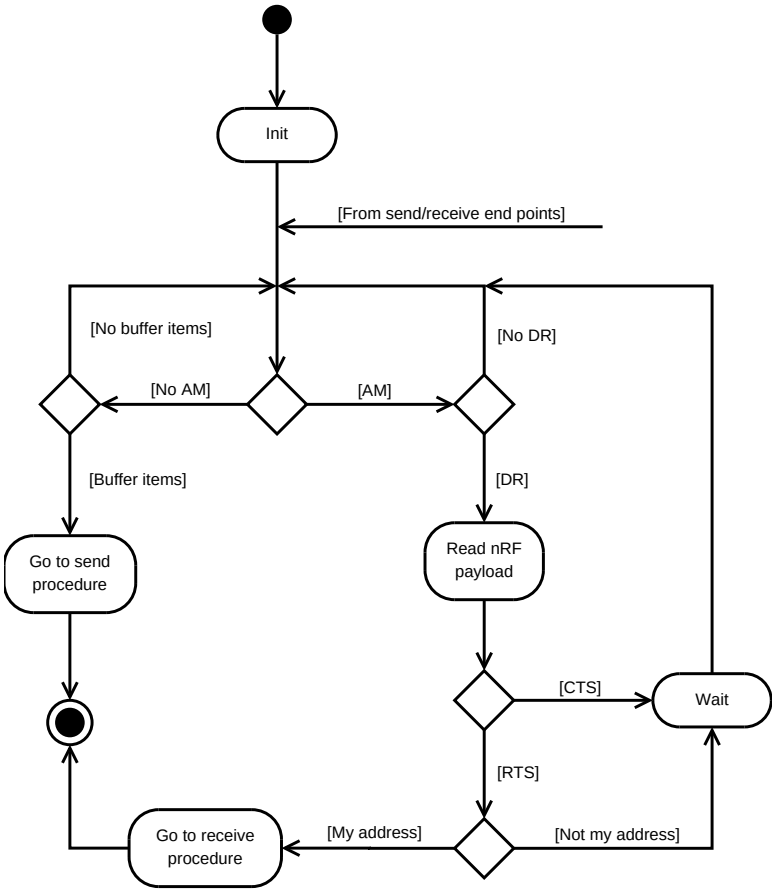


Figure 5.1: *Main activity diagram of the program*

1. The subsystems are initialized
2. If the buffer is empty the mote will continue searching for AM
3. If there are packets in the buffer the mote will start sending
4. If AM occurs it will wait for DR
5. The nRF payload is read

6. If the packet is a CTS, other motes want to send and the mote will backoff (wait)

7. If the packet is an RTS, the mote will start receiving or backoff

When the main flow ends the mote will start either the send procedure or the receive procedure which are described in Sections 5.1.2 and 5.1.3, respectively. The incoming arrow below the Init procedure of Figure 5.1 is from the ends of the sender and receiver flow charts.

5.1.2 Sender flow

Figure 5.2 shows the flow chart of the send procedure of the program. This procedure starts when there are packets in the buffer to be sent to other motes. The main flow chart in Figure 5.1 runs this procedure in the action "Go to send procedure". The send procedure will return to the main flow if a carrier is detected because it could be an incoming packet for the mote. Otherwise the mote will follow the protocol sending RTS, expecting CTS and sending the data packet. Carrier sensing and timeouts are built into these actions to minimize collisions and handle them at occurrence. The actions in the activity diagram are described below.

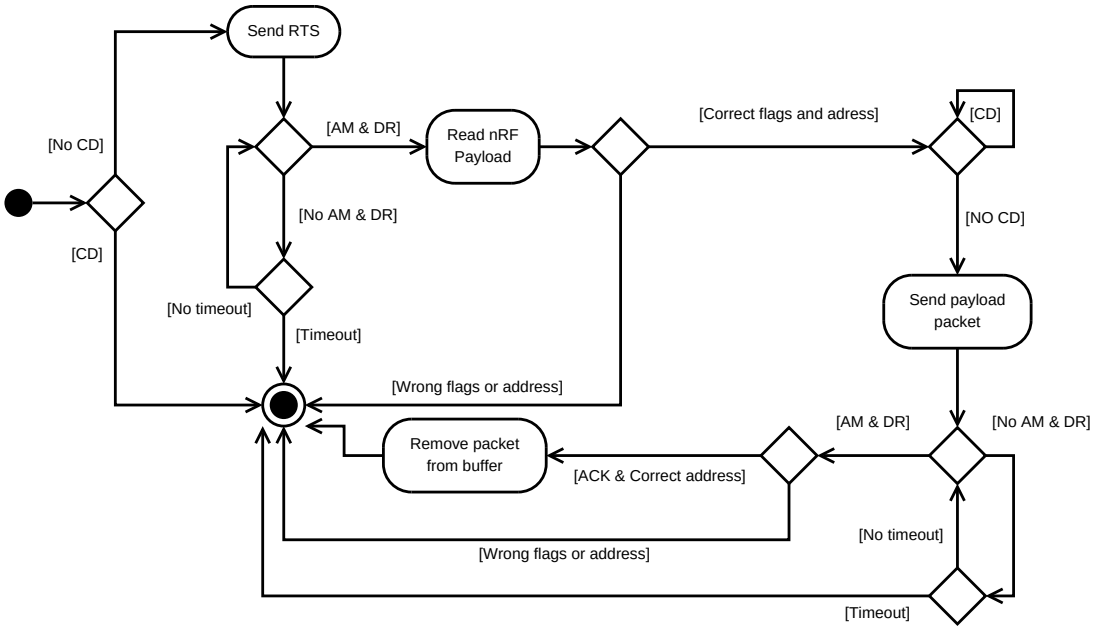


Figure 5.2: *Activity diagram of the send procedure of the program*

1. Before any attempts to send are made, the carrier must be sensed.

2. If the carrier is busy the flow terminates and returns to the main flow, otherwise an RTS packet is sent.

3. The program expects a CTS packet now and waits for AM and DR until they occur or until timeout
4. If a packet is received, it is read from the nRF and verified that it is in fact a CTS packet.
5. Another carrier sensing is made before the data packet is sent.
6. The program waits for an acknowledgement or a timeout.
7. If the timeout occurs the packet must be retransmitted.
8. When a correct acknowledgement is received the transmission is successful. The packet is removed from the buffer and the program returns to the main flow.

If the role of the mote is "output" the program must forward the packets to a mobile phone and not to another mote. In this case the send procedure is different and more simple because Bluetooth handles the multiple access control which allows the mote to see the Bluetooth link as a serial connection. The dsPIC is only connected to the RX and TX pins of the Bluetooth module and can not know if the Bluetooth connection is established or not. Thus a connection flag is needed to indicate the Bluetooth connection status.

The diagram in Figure 5.3 shows the flow of the Bluetooth send procedure. The following steps are executed:

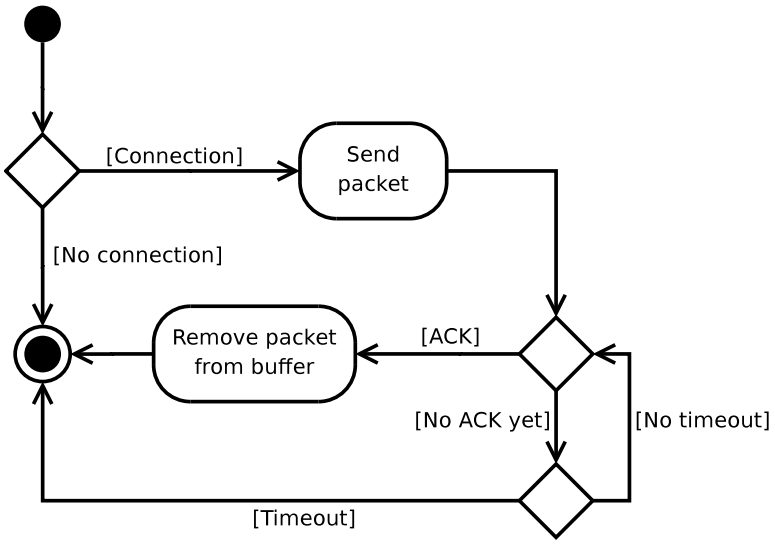


Figure 5.3: *Activity diagram of the send procedure for the output mote of the network.*

1. Check the connection flag to determine if connection is established. Return if no connection.

2. Send the payload of the packet to the mobile phone.

3. Wait for ACK from the phone or return if timeout.

4. When ACK is received, remove the packet from the buffer and return.

5.1.3 Receiver flow

Figure 5.4 shows the flow chart of the receive procedure of the program. This procedure starts when an RTS packet with the correct address is received. It is run from the main flow of Figure 5.1 with the action "Go to receive procedure". The procedure will send a CTS when the carrier is idle and wait for data packet to arrive. If a correct data packet is received, an acknowledgement will be sent. The actions of the receive procedure are described below.

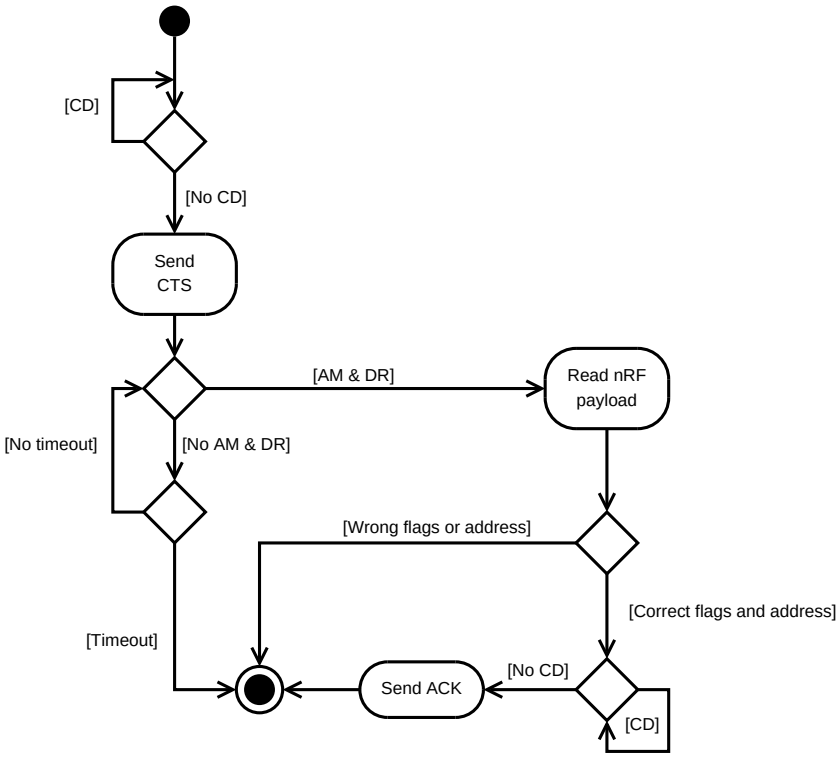


Figure 5.4: *Activity diagram of the receive procedure of the program*

1. A CTS packet is sent when the carrier is idle.

2. The program waits for AM and DR or returns in case of timeout.

3. At AM and DR the nRF payload is read and the packet is checked for correctness.

4. If the packet is correct an acknowledgement will be sent after a carrier sensing.

5. The procedure returns to the main flow.

If the role of the mote is "input", it must receive packets from a mobile phone and forward them to the next mote en the network. As the Bluetooth link to the phone is just a serial connection, the flow is more simple than the general one. Incoming data on Bluetooth (UART 1) is handled by interrupt on the dsPIC. This means that this receive procedure can be run anytime from the main flow and not only when the general receive procedure is run. Figure 5.5 shows the flow of the Bluetooth receive procedure. The following tasks are executed:

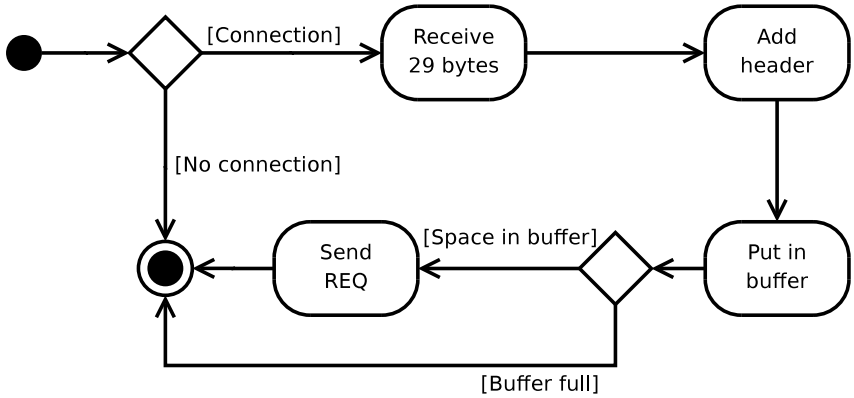


Figure 5.5: *Activity diagram of the receive procedure of the input mote.*

1. Check the connection flag. Return if the connection is not established.

2. Wait until 29 bytes of payload is received. (Interrupt is run for each received byte)

3. Add a header to the payload with sender and receiver address plus the payload flag.

4. Put the packet in the buffer.

5. If there is more space in the buffer, an REQ message is sent to the mobile phone.

6. If the buffer is full, the REQ message will be sent the next time a packet is removed from the buffer.

5.2 UART/BT module

UART1 is used as an interface to bridge the communication between the dsPIC processor and the Bluetooth module in the mote.UART2 is connected to the USB interface and is used for configuration and debugging of the mote program.

To the dsPIC, the UART appears as an 8-bit input and output port that it can read from and write to.

A UART consists of a transmitter, which transmits serial data via a transmit data (TxD) pin, and a receiver, which receives serial data via a receive data (RxD) pin.

5.2.1 Initialization

Before transmitting or receiving data, the UART module must be configured in the following way:

- Set the number of data bits, number of stop bits and parity selection.
- Write appropriate baud rate value to the BRGx register which controls the period of a free-running 16-bit timer. The equation below shows the formula for computation of the baud rate.

$$BRGx = \frac{FCY}{16 * BaudRate} - 1$$

  where FCY denotes the instruction cycle clock frequency.
- Set up receive interrupt enable and priority bits.

Both UART1 and UART2 are configured in the following way:

- 8 data bits
- 1 stop bit
- No parity
- 57600 bit/s
- Receive interrupt enabled, priority 2

5.2.2 Module functions

BT put char

Whenever the mote has data to send to the mobile phone, it just sends it to the UART in byte format (8 bits). The transmission starts only when the data to be transmitted is written to the buffer.

A transmit routine, with a string of characters as an input, will be called each time data is sent from the other nodes to the mobile phone. For this purpose, the routine should first write the string of data to be transmitted into the UART transmit buffer and wait for transmission to complete successfully.

Receive interrupt

Whenever the UART receives data from the mobile phone, it will store it in its FIFO buffer (8 bits), then it will indicate the availability of this data to the mote through an internal register bit. A receive interrupt will be generated when one or more data characters have been received. Thus, the UART module in the reception mode must perform the following tasks:

- Calling interrupt for each byte.
- Read the 29 arrived bytes from the UART data register.
- Write the 29 bytes to a buffer.
- Clear interrupt flag.

USB send

This function is only need for debugging. It will write the input text string to the USB interface to inform about various actions in the program. E.g. reception and transmission of packets.

5.3 Timer module

The purpose of the timer module is to provide the timing functions of the program. The timer is used to set timeouts and to put the program into a wait mode e.g. when other motes has reserved the medium with RTS/CTS packets. There are nine 16 bit timers in the dsPIC. Timer1 is both used to put the running program in a paused state and setting timeout for CTS and ACK packets. The same timer can be used for these events as they never occur at the same time. A timeout for the Bluetooth connection can occur independent of other timeouts, thus Timer2 is used for this purpose.

5.3.1 Initialization

Timer1 and Timer2 is configured in the same way with the following settings:

- Gate time accumulation disabled
- Continue timer operations in CPU idle mode
- Timer prescaler 1:1
- Use internal clock source (same one used by CPU)
- No synchronization of external clock

5.3.2 Module functions

Pause

This function puts the running program in a paused state for a specified duration of time. The input of this function is the duration specified in milliseconds as an integer value n. The function will start the timer and enable interrupt to occur every time the timer reaches a specific value which is the timer period. The clock frequency of the on board oscillator is 22.118.000 Hz which means that the period must be 22.118 clock cycles to make an interrupt each millisecond. The function will return and the program continue when the interrupt routine has run n times.

The Pause function will execute the following tasks:

1. Enable interrupt for Timer1 with highest priority to obtain optimal timing
2. Reset Timer1
3. Start Timer1
4. Wait until n interrupts has occurred
5. Reset number of occurred interrupts
6. Stop Timer1

Timeout

This function is used to wait for expected control packets i.e. CTS and ACK. The function works in the same way as the Pause function, but it will return either when the input time n is exceeded or when a packet is received. Different values will be returned depending on whether the result was timeout or received packet.

BT TimeOut

This function is the same as the Timeout function, but it returns either when the input time n is exceeded or when and ACK is received via Bluetooth.

Timer1 interrupt

This interrupt routine is called every time Timer1 reaches its period value and with the settings of Timer1 this occurs each millisecond. The tasks of this interrupt is the following:

1. Increment number of occurred interrupts
2. Reset Timer1

Timer2 interrupt

Like Timer1 interrupt, this interrupt is called ever millisecond when enabled. It is used to terminate the Bluetooth connection at the input mote when a transmission is complete. This

is done by timeout as stated in Section 4.2.2. The interrupt routine checks if the connection timeout is exceeded to reset a connection flag. This is done in the following tasks:

1. Increment connection time (number of occurred interrupts)
2. If connection time has exceeded the specified time, the connection flag is cleared and the connection time reset.
3. Reset Timer2

5.4 Packet handling module

This module must take care of the incoming packets. This is done by adding or changing the fields in the header i.e. addresses and flags The module is also responsible for creating control packets (RTS/CTS/ACK).

5.4.1 Initialization

No initialization is needed for this module

5.4.2 Module functions

Add header

This function will add the header to the payload packet when it enters the mote network. The function can have the following input:

- Sender address - address of local mote
- Receiver address - address of next mote in the network
- Header flags - RTS, CTS, ACK, PAYLOAD, buffer full and sequence number

Read header

When a mote inside the network receives a packet it must read the content of the header. The function will return the following result:

- Sender address - address of the sending mote
- Receiver address - should be equal to the address of local mote
- Header flags - RTS, CTS, ACK, PAYLOAD, buffer full and sequence number

Put in buffer

When a mote receives a correct payload packet it must change the header such that the packet can be forwarded to the next mote in the network. The function can change the following options:

- Sender address - will be changed to address of local mote
- Receiver address - will be changed to address of next mote in the network

When this is done the packet is put into the buffer and if the buffer is full afterwards a buffer full flag is set. Two pointers (input/output) are controlling where in the buffer the next packet should be saved to and loaded from. It means that the input pointer is pointing on the next empty slot and the output pointer is pointing on the packet which should be selected for next transmission. This is illustrated in Figure 5.6.

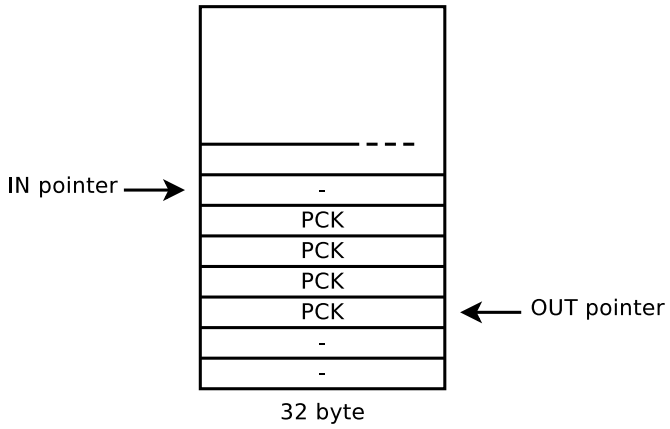


Figure 5.6: *The figure shows the buffer containing four packets and the pointers assigned to it*

Load from buffer

This function will load the packet pointed to by the output pointer in the buffer and return this packet.

Remove header

Before forwarding the packet to the receiver mobile phone the mote header must be removed.

5.5 SPI/nRF module

The dsPIC of the mote is connected to the nRF through one of the SPIs and the nRF is again connected to the loop antenna. This section will describe the setup and the data exchange between the dsPIC and the nRF. The design is partly based on the specifications from the data

sheets of the dsPIC and the nRF, [3] and [8] respectively, because the usage of the on-chip features is strictly specified.

5.5.1 Initialization

The setup of the nRF is done in the 10 byte configuration register. This register and others can be manipulated by writing commands to the SPI of the nRF. The nRF is supporting commands for reading and writing the TX address, TX payload, RX payload and configuration registers. The following states the settings of the configuration register suited for the desired protocol:

Center frequency is set to 433 MHz since it is within the ISM band and the optimal frequency for the antenna.

Output power can take on four different values and will be varied according to an optimal test scenario. A test has been made to measure the transmission range at different transmission powers and initially it is set to the lowest power (-10 dBm) which means a transmitting range of approximately 40 cm. A test of this can be seen in Appendix E.

Reduced receiver sensitivity is set to on to get the wanted max transmission range of approximately 40 cm.

Address width both RX and TX will be the maximum of four bytes to minimize the risk of repeating the address in the first bytes of the payload.[8]

Payload width varies according to the packet type.

- Payload packets: 32 bytes (maximum payload width)
- RTS/CTS/ACK packets: 3 bytes (sender address, receiver address and flags)

RX address is set to the same for every mote. The address is defined to be `MOTE`.

Crystal oscillator frequency is set to 16 MHz corresponding to the external crystal on the mote.

CRC is enabled and set to maximum check bits (16 bits) to minimize errors

5.5.2 Module functions

SPI write and read

The purpose of this function should be to read and write an input character from and to the SPI port, the function will be used to:

- Write nRF configuration

- Write nRF TX address
- Write nRF TX payload
- Read nRF RX payload

The routine of the function must have one of these commands, clock data in/out of the SPI buffer and return data if data is read.

Transmit packet

When data is ready to be sent, the function must do the following:

1. Put radio in standby mode
2. Put the nRF module into transmit mode
3. Send "Write nRF TX payload" to the nRF
4. Call the SPI write/read function in loop to clock data into the nRF TX payload register
5. Enable the nRF for transmission
6. Wait until transmission is finished
7. Put the radio in enable mode

Receive packet

When data is ready to be received from the nRF, the function must do the following:

1. Put radio in standby mode
2. Send "Read nRF RX payload" to the nRF
3. Call the SPI write/read function in loop to clock data out of nRF
4. Save output in a variable
5. Put the radio in enable mode

Chapter 6

Mobile application design

In this chapter the design of the application for the mobile phone will be described. The chapter is split into five sections: One that describes the general design of the application, two for describing the sender and the receiver applications, one that describes the user interface design and one describing the module design.

6.1 General description

The main purpose of the application is to allow a user to send or receive data to or from a mote network. In order to fulfill this purpose and the requirements specified in Section 3.4, the application must have two different functionalities:

- A sender used to send data into the network
- A receiver used to receive data from the sender through the network.

These two functionalities correspond to two different programs which are designed separately. The two different programs will run in the sender and the receiver phones, respectively.

6.1.1 File types

In order to test the mote network with traffic types that can be commonly used in real applications, it has been chosen to design a program allowing the user to send and receive either:

- Text files (**txt**)
- Image files (**jpg**)
- Audio files (**wav**)

These files are sent through the mote network in a non-real time channel. The program should therefore implement functions for creating/reading all these three kinds of files.

6.1.2 Protocol

Once a file is created, it will be divided into smaller packets, each of which must have a size of 29 bytes according to Section 3.4, that will be sent into the network one by one.

The communication between the mobile phone and the motes is a Bluetooth serial connection. Error correction is already implemented in the Bluetooth standard and acknowledgements for each sent packet are automatically forwarded from the Bluetooth interface of the mote to the mobile phone [9].

This means that when a packet is sent, the next packet will be sent only when the first one is correctly received. The program must however handle the data flow control in order not to overfill the buffer on the mote. In order to fulfill this purpose, the program will only send one packet to the mote, every time the mote will request it. Figure 4.5 shows the steps involved in the transmission of each packet. After a file is created, a first packet is sent to the receiving mote containing:

- The extension of the file (txt, jpg or wav)
- The size of the file in bytes

The first three bytes contain the extension as a text string. The size field is variable and the sender has to contain a module to fill out the first packet to the standard size of 29 bytes. Thus some additional bytes are added to fill up the packet. The purpose of sending this information is to inform the receiving mobile phone about the size of the file, and therefore the number of packets it must receive and the way it has to interpret them.

After the first packet, the other packets sent after each request will only contain the data of the file without any additional information. In order to obtain a higher data rate, it has been chosen to use only the first packet for transmitting the size and the type of the file, rather than a header for each packet sent. This makes the protocol less general and more specific for the network. Figure 6.1 shows the differences between the two methods.

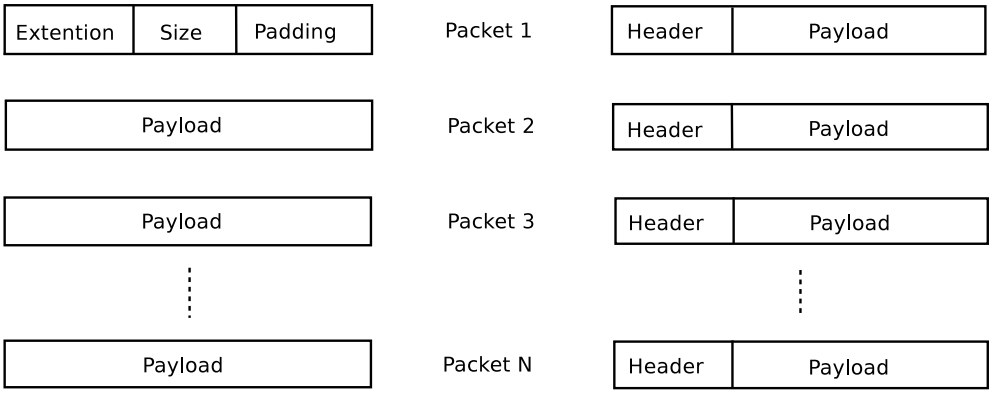


Figure 6.1: *Two methods for the mobile phone protocol. To the left, the first packet acts like a header. To the right, every packet has a header.*

6.2 Sender

The sender device must be able to send data via Bluetooth to the first mote on the network. Initially it has to create or load a file. The user can choose the type of the file which is going to be sent. The choices are:

- Text
  - **Write:** An editor will appear allowing the user to write some text
  - **Load:** A specific text file saved earlier in a folder of the phone will be sent
- Picture
  - **Take:** The user will be able to use the camera to take a picture
  - **Load:** A specific picture saved earlier in a folder of the phone will be sent
- Audio
  - **Record:** A voice recorder will run
  - **Load:** A specific audio file saved earlier in a folder of the phone will be sent

The phone searches for all the reachable Bluetooth devices and displays them on a list. Afterwards it establishes a connection via Bluetooth with the mote selected by the user from the list. In order to inform the mote about its presence, the mobile phone will initially send an initialization packet.

Each file, before being sent, has to be split in more packets, whose length is 29 bytes each. The first and the last packet could contain less than 29 bytes and it is necessary to fill up these packets with a sequence of bytes until the length reaches 29 bytes. This is done by a padding function.

After establishing the connection with the mote the mobile phone will send the first packet containing the information about the size and the type of the file. Every time the mote receives a packet, including the initialization message, it will send a request packet which allows the phone to send another packet.

When the file is completely sent the phone closes the connection. Figure 6.2 describes the general working of the sender program.

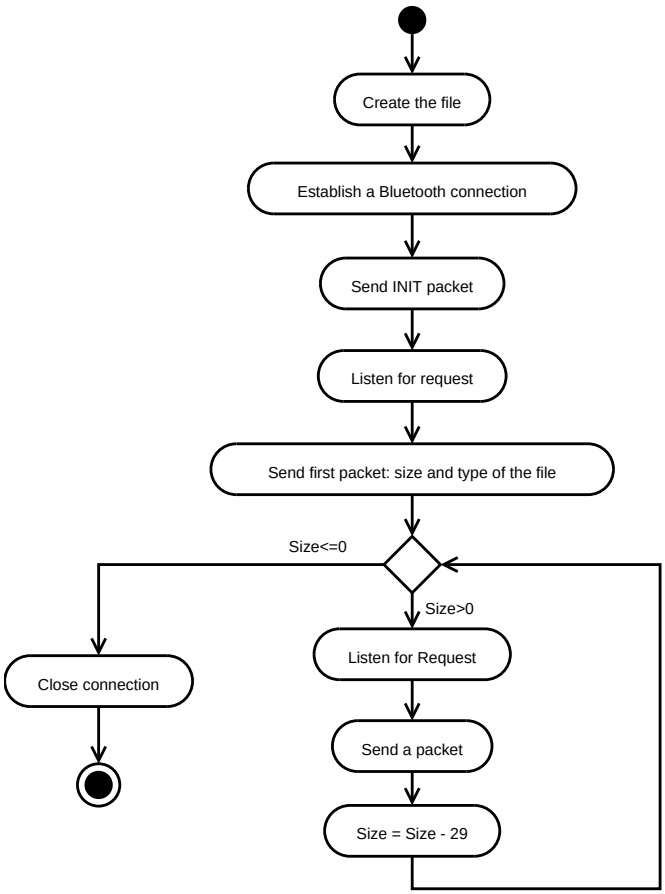


Figure 6.2: *The figure shows an activity diagram of the sender program.*

Figure 6.3 shows the steps involved in the creation of a file.

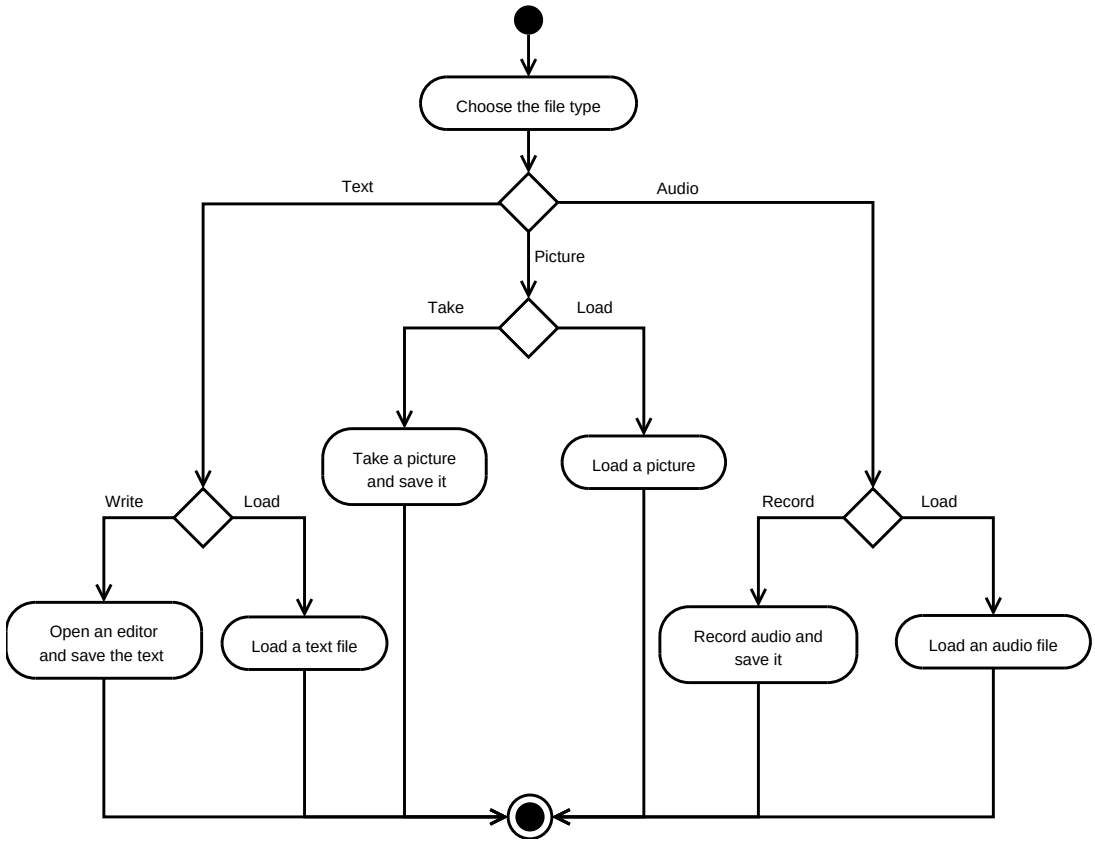


Figure 6.3: *The figure shows an activity diagram of the procedure for creating a file.*

6.3 Receiver

The receiver phone has to establish the Bluetooth connection with the last mote in the network because the Bluetooth module in the mote is not able to establish the connection by itself. When the connection is established an initialization packet is sent. Afterwards the phone starts listening. The first received packet contains the information about the size and the type of data which is stored in two variables.

A loop keeps running as long as the size variable is larger than zero and the mobile phone will send an acknowledgement to the mote for each packet received.

When the loop ends, the phone saves the file received, displays or plays it and closes the connection after sending an end message. This message is sent to inform the network about the closing of the connection. Figure 6.4 shows the main operations performed by the receiver program.

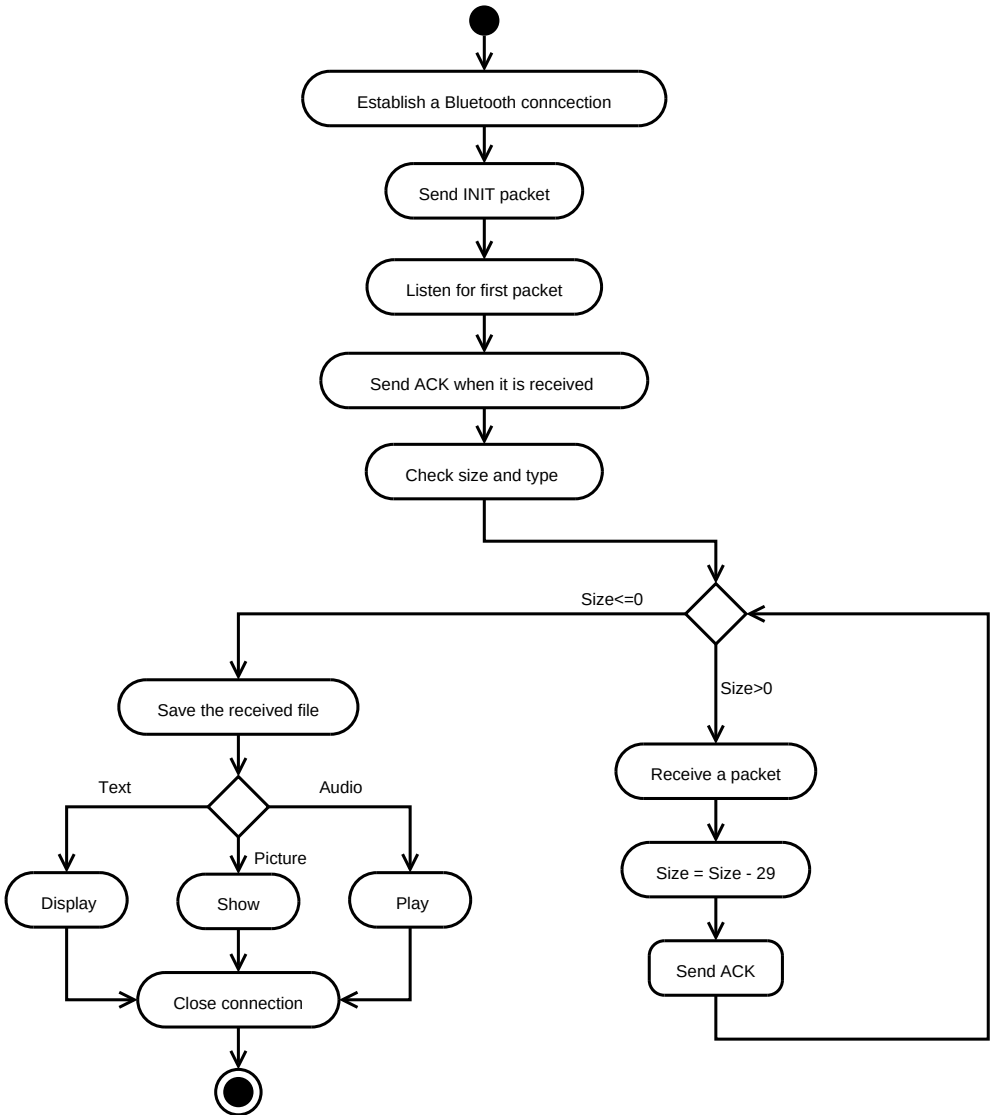


Figure 6.4: *The figure shows an activity diagram of the receiver.*

6.4 Graphical user interface

This section contains the design of the sender and the receiver GUI. Each of the programs is structured as a menu the user can navigate through.

6.4.1 Sender application

When the sender application is started, a menu will be opened. As it is shown in Figure 6.5 the user is able to choose which kind of file he wants to send and select to create or load it.

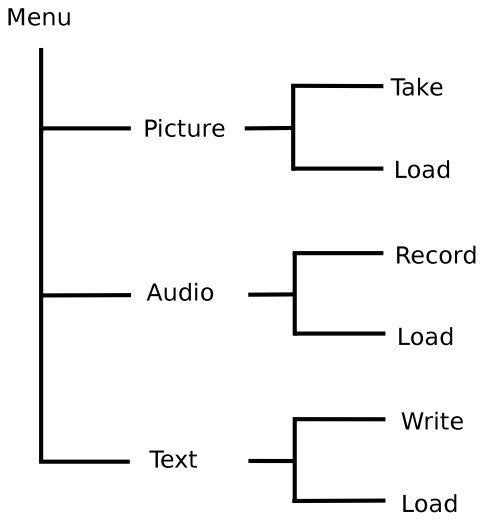


Figure 6.5: *The figure shows the structure of the menu*

After creating or loading the file, the program will start to search for Bluetooth devices in the surrounding area, and it will display them in a list. The user will interact with the program by choosing a displayed device, and when the connection will be established a confirming message, and the address of the receiver device will be printed. The file is therefore divided in packets and started to be sent. When it is completely sent a confirming message will be displayed.

When the file is completely sent the program will close the connection with the mote, and the starting menu will appear again, allowing the user to create/load a file again and to send it.

6.4.2 Receiver application

When the receiver application is started, and the "search for devices" command is chosen, a list with all the Bluetooth devices reachable is displayed until the user will select one. Therefore when the connection is correctly established, the user will be informed about the status of the program by a message saying "I'm listening". When the mobile phone starts to receive a file, a message saying "I'm receiving" appears in the screen as long as the whole file is not received.

Depending in which kind of file the mobile phone has received it will:

- Display the message
- Show the picture
- Play the audio file and display a "Playing" message

6.5 Modules

This section deals with the description of the main modules the application will contain. The following subsections will describe the modules used for establishing the Bluetooth connection, for the sender application, for the receiver application and the creation of the GUI. A module in the mobile phone application is defined as a function executing a task. The modules are listed and described in different categories.

6.5.1 Bluetooth connection modules

- **Socket opener:** This module will be used to open a socket.
- **Devices searcher:** This module will be used to search for other Bluetooth devices in the area.
- **Connect:** This module will be used to connect to one of the discovered devices.
- **Connection closer:** This module will close the connection, when it is no more needed.

6.5.2 Sender modules

- **Text writer:** This module creates a text file and save it in a specific folder
- **Text loader:** This module loads a text file from a specific folder in the phone
- **Camera opener:** This module starts the camera
- **Picture taker:** This module takes a picture and save it in a specific folder
- **Picture loader:** This module loads a picture from a specific folder in the phone
- **Audio recorder:** This module records an audio file and save it in a specific folder
- **Audio loader:** This module loads an audio file from a specific folder in the phone
- **Size checker:** This module checks the size of the created or loaded file
- **Extension checker:** This module checks the extension of the created or loaded file
- **File splitter:** This module splits the file in packets of 29 bytes
- **Padder:** This module fills up the the first and the last packet, that can have less than 29 bytes of data
- **Init sender:** This module sends an initialization message before starting the communication
- **Send:** This module will send a packet every time it is called

- **Listen:** This module will put the phone in a waiting status until a request for packets is received from the mote to which the phone communicate via Bluetooth

6.5.3 Receiver modules

- **Init sender:** This module sends an initialization message before starting the communication
- **Listen:** This module will put the phone in a waiting status until a packet is received from the mote
- **Send:** This module will send an acknowledgement packet for each data packet received
- **File maker:** This module joins all the packets received in a unique file
- **File recognizer:** This module recognizes the type of the file received
- **Text reader:** This module displays the text received
- **Picture viewer:** This module displays the picture received
- **Audio player:** This module plays the audio file received

6.5.4 GUI modules

- **Menu creator:** This module creates the start menu which is used by the user to choose which type of file to be sent and whether to create it or load it from a specific folder
- **Connection status:** This module displays a short messages which informs the user about the opening/closing of the connection, the file transfer and the errors
- **Picture viewer:** This module opens a canvas in which the received picture is appended

Chapter 7

Implementation

The purpose of this chapter is to describe the difference between the design and the implementation. It is split into two sections describing the mote protocol and the mobile phone application. The programming language used to implement the MAC protocol on the dsPIC is C, while the mobile phones are programmed in Python. First all the changes made to get the software operating properly are illustrated. A command line interface is implemented to configure the mote via the USB interface. Then, to give a better overview of how the motes react in the environment, some debug information are sent to the USB to be printed in a terminal program. Also interactions between the mote network and the sender/receiver mobile phones can be seen in screenshots. The source code of this project is not discussed in chapter, but it can be found on the attached CD.

7.1 Mote protocol

In the implementation of the protocol on the motes some problems where encountered in receiving acknowledgement and termination commands from the mobile phone. This problem was solved by correcting the timing between the mobile phone and the mote so the mobile phone only would send ACK and END when it had received exactly 29 bytes of data. The commands "ACK" and "END" was also changed to "A" and "E" to ease the communication flow.

7.1.1 Command line interface

The following commands create a user friendly interface to the mote allowing the user to access and to reconfigure the different features in the mote program.

put *string* : The message which comes after the put command is stored in the buffer to be forwarded into the network.

buf : This command will print the number of packets contained in the buffer and print the payload of the next outgoing packet. If the buffer is empty, a message will be displayed.

chaddr *char* : This command is used to change the current mote address to any desired character.

moteset : This command print the mote address, its neighbors as well as its role in the network: I, O or F.

ISM : To view the nRF settings: center frequency, frequency band and power, TX and RX address and payload width, nRF address, CRC and clock, this command will print the hexadecimal values for each byte in the nRF configuration register.

role *char* : If this command is typed, it will change the mote roles in the network to I, O or F.

neighbors *char char* : This command is used to set the neighbors of the mote.

7.1.2 Debug information

The debug information displayed in Listings 7.1 and 7.2 are the output of the two end motes in the network.

First, an Init message is sent initializing the connections: Sender mobile phone - input mote and receiver mobile phone - output mote. The first message sent by the mobile phone into the mote network is containing information about the type and the size of the payload data. Then, the packet is sent. The first mote is transmitting an RTS message successfully received by the last mote which responds with a CTS. The output mote then receives the payload packet, stores it in the buffer and send an ACK back. In the meantime, the output mote will also receive an ACK from the receiver mobile phone proving that the packet was received correctly.

To convey the payload through the network, the first mote has to send again an RTS, but collisions occur, so the RTS is retransmit, three times in this case, until is received by the last mote. The payload packet is then conveyed through the network, using the same scheme as before.

A timeout occurs, meaning that there are no more packets to send, so the first mote close the Bluetooth connection with the sender mobile phone. When the receiver mobile phone has all the packets, instead of sending an ACK to the last mote, it will send an END message closing the connection.

Note that the motes are not allowed to read the payload.

Listing 7.1: *Debug information for the input mote*

```
Init received
RTS header sent
```

```
Receive packet: Source: 2 Destination: 1 Type:CTS
Source: 2 Destination: 1 Type:ACK
Payload packet was sent

RTS header sent
RTS header sent
RTS header sent
Receive packet: Source: 2 Destination: 1 Type:CTS
Source: 2 Destination: 1 Type:ACK
Payload packet was sent

Bluetooth connection is closed
```

Listing 7.2: *Debug information for the output mote*

```
Init received

Receive packet: Source: 1 Destination: 2 Type:RTS
CTS header sent
Source: 1 Destination: 2 Type:PAY
To put in buffer: Source: 1 Destination: 2 Type:PAY
New destination address: Source: 1 Destination: M Type:PAY
Put in buffer: Source: 2 Destination: M Type:PAY
ACK header sent
Received ACK from mobile phone

Receive packet: Source: 1 Destination: 2 Type:RTS
CTS header sent
Source: 1 Destination: 2 Type:PAY
To put in buffer: Source: 1 Destination: 2 Type:PAY
New destination address: Source: 1 Destination: M Type:PAY
Put in buffer: Source: 2 Destination: M Type:PAY
ACK header sent

Mobile phone closed the connection
```

7.1.3 Protocol timing

The designed protocol states that a mote must backoff/wait when it overhears RTS and CTS between other motes. The design in Chapter 5 however, does not specify how long time the

backoff should last. This section will describe the practical and theoretical determination of the backoff time. The backoff time should be slightly different for a received RTS compared to a CTS, but for simplicity it is chosen to use the same backoff time in both cases.

The backoff time to follow an overheard RTS must be the time to send: CTS, Payload and ACK. To calculate total time the following timing results is needed:

- Time to send RTS/CTS/ACK packet (3 bytes): $8bits/byte \cdot 3bytes \cdot 10^{-5}s/bit = 0,24ms$
- Time to send payload packet (32 bytes): $8bits/byte \cdot 32bytes \cdot 10^{-5}s/bit = 2,56ms$
- Time for nRF RX/TX mode switch: $0,55ms$ [8]
- Time for standby/power up switch (The nRF must be in standby to receive new payload via SPI) : $0,65ms$ [8]

The actions needed to complete a transmission after a sent RTS is (data transmission between nRF and dsPIC not included):

1. Switch to RX mode
2. Receive CTS packet
3. Switch to standby mode
4. Switch to TX mode
5. Send payload packet
6. Switch to RX mode
7. Receive ACK packet

The total theoretical backoff time can now be calculated:

$BackoffTime_{theoretical} = 0,55 + 0,24 + 0,65 + 0,55 + 2,56 + 0,55 + 0,24 = 5,34$

In practice, there is additional issues to be considered e.g. the data transmission between nRF and dsPIC, and the processing time in the dsPIC. This is assumed to be very fast ($< 1ms$) and is of minor importance. The mote program is designed to do a carrier sense before each transmission and this takes a random amount of time which is hard to estimate, but the calculated theoretical time is believed to a qualified guess. This makes a total backoff time of approximately $10ms$.

Testing of the implementation in the development phase proved that $10ms$ was to short to avoid collisions. The increased transmission time is caused by the printing of the debug information shown in Listings 7.1 and 7.2. With the chosen baud rate of $57600bit/s$, the time to transmit one character is:

$\frac{1}{57600} s/bit \cdot 8bit/char = 0,14ms$

At every received packet, approximately 40 characters will be printed via USB. In the case of 40 characters, an additional $40 \cdot 0.14ms = 5.6ms$ must be added. This leaves a practical backoff time of approximately $20ms$ which works good in the implementation.

7.2 Mobile phone application

In the implementation of the mobile phone application a few changes have been made to improve the system. It turns out that some garbage characters is written to the UART when the mobile phone establish a connection to the Bluetooth module on the mote. To remove this garbage the mobile phone needs to send a string terminated by \n to flush the UART buffer. The initialization message should then be as follows:

- PREINIT\n
- INIT\n

7.2.1 Screenshots of the UI

The mobile phone application implemented in Python is split into two different parts, a sender and a receiver application. In Figure 7.1 text in selected as desired content and a text message "Hello world!" is written in the UI. Then in Figure 7.2 a search for Bluetooth devices is performed and a connection to the input mote is made and the message is sent to the network. For the receiver side Figure 7.3 shows that the receiver is started and connected to the output mote and afterwards the message is received correctly.

Figure 7.1: *On the left figure the user selects to write a text message, on the right a message is written on the sender mobile phone*

Figure 7.2: *On the left figure a Bluetooth search is performed, on the right the connection is established and the message is sent into the network*

Figure 7.3: *On the left figure the user is starting the receiver program, in the middle a Bluetooth search is performed and on the right the mobile phone is connected to the mote and have received the message "Hello world!"*

Chapter 8

Test

In this chapter different test approaches is described: First the mobile phone applications are tested separately with a simulated network, then it is described how the collision avoidance scheme can be tested and finally the results from the acceptance test is outlined.

8.1 Test of the mobile phone application

In this section the test results from Appendix D is described. It is a test of the mobile phone applications with a PC simulating the mote network. The results of the test can be seen in Table 8.1.

Case	**Test**	**Result**
1	Sending text	Passed
2	Sending image	Passed
3	Sending audio	Passed

Table 8.1: *Results of the test with the mobile applications and simulated mote network*

This shows that both text, image, and audio files have been sent from the first to the second mobile phone through the simulated network and in all cases the file has been correctly received.

8.2 Test of the collision avoidance scheme

The collision avoidance scheme (RTS/CTS) plays a major role in the implemented protocol and therefore it is essential to discuss how it can be tested that the scheme can avoid the hidden terminal problem as well as limiting the numbers of collisions in the network. One way to test

this could be to connect each mote to a PC and view the debug information. To make it possible to debug, the communication flow needs to be delayed (e.g. 5 seconds) so that it is possible to see each operation taking place in the network with a human eye. To test whether the protocol avoids the hidden terminal problem the nodes needs to be placed as it is described in the network scenario in Section 4.1. This test is hard to do because the transmitting/receiving range for the motes is different due to inequality in the calibration of the antennas. This problem could be solved by adjusting the nRF power settings on the motes or by recalibrating the antennas. Because of the limited project period this test is not performed.

The expected results of this test in the scenario of Figure 4.1 with transmission between mote 2 and 3 would be:

1. Mote 2 sending RTS
2. Mote 1 receiving RTS and printing "backoff"
3. Mote 3 receiving RTS and sending CTS
4. Mote 4 receiving CTS and printing "backoff"
5. Mote 2 and 3 exchanges payload and ACK
6. Either mote 2 or 3 will try to send and the neighbors will backoff

8.3 Acceptance test

In this section the results from the acceptance test are discussed. This is done in order to conclude whether or not the system complies with the requirements specification stated in Chapter 3. The acceptance test specification can be seen in Appendix A.

8.3.1 System acceptance test

The system acceptance test is testing the overall system functionality. In Table 8.2 the results from these tests are stated.

Case	Test	Result
1	Number of devices in the system	Passed
2	Bluetooth module in the motes	Passed
3.a	Text transfer	Passed
3.b	Picture transfer	Passed
3.c	Audio transfer	Passed

Table 8.2: *System acceptance test results*

The acceptance tests performed on the overall system verifies that the system complies with the requirements.

8.3.2 Mote protocol acceptance test

The mote protocol acceptance test is testing the different aspects regarding the protocol. Table 8.3 summarize the results from these tests.

Case	Test	Result
1	1-persistent carrier detecting	Passed
2	Collision avoidance (RTS/CTS) scheme	Passed
3	Acknowledgement packets (ACK)	Passed
4	Retransmission of RTS/CTS exchange after ACK timeout	Passed
5	Put packet longer than 29 bytes in a buffer	Passed
6	Header information	Passed
7	Typing ISM	Passed
8	Maximum transmitting range between the motes	Failed
9	Power supply of the motes	Passed
10	Connection between mobile phones and motes	Passed
11	Convey message	Passed
12	Stress test while sending long message	Passed

Table 8.3: *Mote protocol acceptance test results*

The acceptance tests performed on the mote protocol verifies that the protocol almost complies with the requirements. Test case 8 is stating that the motes should be out of range if the

distance between them are above 50 cm. The implemented maximum transmitting range is 40 cm, but this range is varying from mote to mote due to inequality in the calibration of the antennas. This means that one mote is able to transmit or receive at a range of 1 m while another one only is capable of a range of 20 cm.

8.3.3 Mobile phone application acceptance test

This acceptance test is testing the functionality of the mobile phone applications. Table 8.4 summarize the results from these tests.

Case	**Test**	**Result**
1	Mobile phone and mote in range	Passed
2	Sender and receiver applications	Passed
3	Sender menu	Passed
4	Receiver menu	Passed
5	Sender functions for all file types	Passed
6	Receiver functions	Passed
7	Data splitting	Passed

Table 8.4: *Mobile phone application acceptance test results*

The acceptance tests performed on the mobile phone applications verifies that the applications complies with the requirements.

8.3.4 Acceptance test conclusion

The acceptance test has been executed with a satisfying result. The requirements of Chapter 3 is fullfilled except for Requirement 8 to the mote protocol. The reason for this failure is hardware related and can not corrected in the software. Thus the implementation is correct in respect to the requirements specification.

Chapter 9

Conclusion

The main purpose of this project was to implement a MAC protocol for the ISM module of the second version of the mote developed by Aalborg University. This creates a basis of a network which can be used by e.g. mobile phones to exchange data and extend the Bluetooth range.

Through the analysis of different MAC protocols for wireless networks, different channel sharing mechanisms were examined, including static allocation methods like TDMA and FDMA, the Aloha approach, various kinds of carrier sensing protocols and collision avoidance schemes. Some of the problems of the the wireless domain were also investigated i.e. the hidden and exposed terminal problems and it is shown that the collision avoidance schemes can help solving these problems.

Based on the analysis it was chosen to design and implement a MAC protocol that can minimize collisions in the network. Also it is desired to make a robust and simple protocol with no loss of data. For this a collision avoidance scheme is used in combination with acknowledgements and carrier sensing (RTS/CTS/DATA/ACK). The carrier sensing is also needed in the unlicensed ISM band where transmissions between other devices can easily occur. It was also chosen to develop a mobile phone application which is used to exchange text, image and audio files through the mote network implementing the MAC protocol.

The topology of the mote network is chosen to be a line which is a simple topology with fixed routing. This makes it possible to examine the performance of the protocol dealing with the hidden terminal problem. The network supports a unidirectional flow of data from one mobile phone to another. The program for the motes in the network has been designed to fulfill the requirements specification and to run on the given mote hardware. Two programs have been designed for the mobile phone application: A sender program for creating a file to be sent into the network according to the specified mobile to mote interface, and a receiver program to receive and read the transmitted file.

The implementation of the mote program was done in C, while the mobile phone application

was done in Python. Some minor changes were made compared to the design to optimize the performance of the Bluetooth connection between the motes and mobile phones. A command line interface was also implemented for easy debugging and testing purposes. The timing of the collision avoidance scheme has also been calculated and estimated to make an optimum implementation.

The protocol and the mobile phone application were tested with four motes and two mobile phones. By the test it was concluded that the system works, accomplishing all the requirements made. A user can send and receive text, image and audio files through a robust mote network without duplication or loss of packets. One of the requirements did not pass the acceptance test, but the error is in the antenna calibration and is not software related. The avoidance of the hidden terminal problem has not been tested due to the hardware configuration of the mote antennas and limited project period. This should however be possible to do and would optimize the MAC protocol even further.

Chapter 10

Future perspectives

In this chapter ideas and perspectives for the future developments and enhancements for the wireless mote network and MAC protocol will be presented. The chapter is divided into two sections describing enhancements for the protocol and mobile phone application respectively.

Enhancements for the protocol

Connection establishment mote-mobile phone

To be able to receive a message, the mobile phone must initialize a Bluetooth connection with the output mote. In future development the output mote should instead be able to search for the receiver mobile phone and to connect to it. This can be done by programming the Bluetooth module on the motes.

Bidirectional network

When the sender mobile phone wants to transmit a message, the packets are conveyed through the network in an unidirectional way, this means that the sender is not informed about the success/failure of the transmission. By introducing a bidirectional communication flow it will be possible to exchange data both ways. This could enhance the protocol so it will be possible for the receiver to inform the sender about success/failure like in TCP. This could also ensure data integrity because the receiver will be able to send acknowledgements for each packet. By having a bidirectional network it is also possible to have applications like chat or walkie-talkie.

Broken links and dynamic routing

If one mote situated on position M in the network is removed or is put out of the range of its neighbors or simply does not work anymore, the mote in position M-1 will continuously send an RTS message. Instead, the protocol can be improved to solve this problem by sending a message back to the mobile phone informing the last one that there is a problem, and the

message did not reach its destination. Another approach is to solve this problem by introducing dynamic routing schemes. If there is a broken link the protocol should find another route for the packet. The next step to enhance the routing protocol could be to implement a program calculating the best path for each packet.

Add sensor modules to the motes

The motes do not have any sensing capabilities so a future enhancement in the hardware is to add sensor modules to the current motes which can be programmed to collect temperature, humidity, light, pressure, chemical substances, or vibration information, and report it to the mobile phone. In this way the protocol will be suited for direct deployment in a final system.

Enhancements for the mobile phone application

One mobile application containing sender and receiver part

To have a bidirectional network the mobile phone application should be able to send and receive at the same time. To make this possible the application needs to be implemented in one program capable of running both a sender and a receiver thread.

More file types supported

In future development the mobile phone application will be able to send/receive more file types e.g. video files. Mp3 file support could also be implemented to decrease the transmission time of audio files.

GUI

The GUI of the application could also be improved to handle incoming sensor measurements by using pop-up messages.

Bibliography

[1] Dhvanish Chokshi. *Design of a MAC Protocol with Need-Based Scheduling for Wireless Sensor Networks.* Available at: `http://www.csuohio.edu/ece/theses/2005/chokshi.pdf`, 2005.

[2] Phil Karn. *MACA - A New Channel Access Method for Packet Radio.* Available at: `http://people.qualcomm.com/karn/papers/maca.html`, 1990.

[3] Microchip. *dsPIC33F Family Data Sheet.* Available at: `Availableat:http://ww1.microchip.com/downloads/en/DeviceDoc/70165E.pdf`, 2007. Accessed 1/4-2007.

[4] Petar Popovski. *Link layer protocols and multiple access.* Available at: `http://kom.aau.dk/%7Epetarp/Teaching/mm2_F06.pdf`. Accessed 24/4-2007.

[5] Ramjee Prasad. *Universal Wireless Personal Communications.* Artech House, 1998. ISBN: 0-89006-958-1.

[6] Raphael Rom. *Multiple Acces Protocols.* Springer-Verlag, 1989.

[7] Jochen Schiller. *Mobile Communications.* Addison-Wesley, 2000. ISBN: 0-201-39836-2.

[8] Nordic semiconductor. *Product specification. Single chip 433/868/915 MHz Transceiver nRF905.* Available at: `http://www.semiconductorstore.com/pdf/NRF905%20%20%20%20%20%20%20%20%20.pdf`, 2004. Accessed 1/4-2007.

[9] Bluetooth SIG. *Architecture - Data Transport.* `http://bluetooth.com/Bluetooth/Learn/Works/Data_Transport_Architecture.htm`. Accessed 29/5-2007.

[10] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall PTR, 4th edition, 2003. ISBN: 0-13-038488-7.

[11] Scott Shenker Vaduvur Bharghavan, Alan Demers and Lixia Zhang. *MACAW: A Medium Access Protocol for Wireless LAN's.* Available at: `http://pdos.csail.mit.edu/decouto/papers/bharghavan94.pdf`, 1994.

[12] John Heidemann Wei Ye and Deborah Estrin. *An Energy-Efficient MAC Protocol for Wireless Sensor Networks*. Available at: `http://www.isi.edu/~weiye/pub/smac_infocom.pdf`, 2002.

[13] Wikipedia. *ALOHAnet*. `http://en.wikipedia.org/wiki/ALOHAnet`. Accessed 24/4-2007.

Appendix A

Acceptance test specification

The acceptance test specification describes how to verify the requirements specification in Chapter 3. The test specification states a case for verifying each requirement. The cases are ordered under the same sections as in the requirements specification and the case number corresponds to the requirement number.

System requirements

Case 1

Actions: Count the number of motes and mobile phones in the system.

Success criteria: There are 4 or more motes and 2 mobile phones.

Case 2

Actions: Verify that 2 of the motes have a Bluetooth module mounted on them.

Success criteria: 2 and only 2 of the motes have Bluetooth modules.

Case 3.a

Actions:

1. Verify that mobile phones and motes are activated.
2. Start the application on each mobile phone and establish a connection to a mote from each phone.

3. Send a text message from the sender phone.

Success criteria: The message from the sender phone is received correctly at the receiver phone.

Case 3.b

Actions:

1. Verify that mobile phones and motes are activated.
2. Start the application on each mobile phone and establish a connection to a mote from each phone.
3. Send a image file from the sender phone.

Success criteria: The file from the sender phone is received correctly at the receiver phone.

Case 3.c

Actions:

1. Verify that mobile phones and motes are activated.
2. Start the application on each mobile phone and establish a connection to a mote from each phone.
3. Send a audio file from the sender phone.

Success criteria: The file from the sender phone is received correctly at the receiver phone.

Mote and protocol requirements

To verify the requirements to the protocol, a debug functionality must be implemented. This must allow the tester to see what is happening on each mote e.g. as text messages printed through the USB interface on the motes. In the following test cases, it is assumed that the motes are activated and running the protocol program. When referring to print results, additional debugging information may be printed as well.

Case 1

Actions: Put a packet in the buffer of a mote either via the USB interface or the mobile phone application.

Success criteria:

1. "Packet put in buffer" is printed.
2. "Sending packet" is printed.
3. "CD" (Carrier Detect) is printed a random number of times (maybe even zero times) to indicate the continuous carrier sensing.
4. "Sending packet" is printed.

Case 2

Actions: Put a packet in the buffer of a mote either via USB interface or mobile phone application.

Success criteria:

1. "Packet put in buffer" is printed.
2. "RTS sent" is printed with source and destination address.
3. "CTS received" is printed with source and destination address reversed.

Case 3

Actions: Put a packet in the buffer of a mote either via USB interface or mobile phone application.

Success criteria:

1. "Payload sent" is printed.
2. "ACK received" or "ACK timeout. Sending again" is printed.
3. If "ACK received" the payload packet must be in the buffer of the receiver mote.

Case 4

Retransmission of RTS/CTS exchange after ACK timeout.

Actions:

1. Put a packet in the buffer of a mote.

2. "RTS sent" and "CTS received" is printed.

3. Move the receiver mote out of the transmission range of the sender mote.

Success criteria:

1. "Payload sent" is printed

2. "ACK timeout. Sending again" is printed 3 times

3. "RTS sent" is printed again

Case 5

Actions: Put a packet in the buffer via the USB interface as a string containing more than 29 characters.

Success criteria: Only the first 29 characters are sent.

Case 6

Actions: Put a packet in the buffer of one mote via the USB interface, while the others are deactivated, and type "buf".

Success criteria: The packet is shown with 3 bytes of header information.

Case 7

Actions: Type "ISM" via the USB interface.

Success criteria: The ISM interface name is printed as 4 characters (= 32 bits).

Case 8

Actions:

1. Move a receiving mote more than 50 cm away from the sending mote.

2. Put a packet in the buffer of a the sending mote.

Success criteria: "RTS sent" is printed continuously, meaning out of range.

Case 9

Actions: Verify that the motes are connected to a power supply.

Success criteria: The power supply of the motes is either a battery or a static power supply e.g. USB connection to a laptop.

Case 10

Actions: Use the mobile phones application to connect to a Bluetooth mote.

Success criteria: "Init received" is printed via the USB interface of the mote.

Case 11

Actions: Send a message from the mobile phones through the network.

Success criteria: The file is received correctly and can be displayed to the user.

Case 12

Actions:

1. Send a long text message (e.g. 1000 characters) with different sentences.
2. While transmitting, move the motes randomly in and out of range of its neighbors.
3. Wait for transmission to finish.

Success criteria: The correct message is shown on the receiver mobile phone without any redundant sentences.

Mobile phone application requirements

Case 1

Actions: Search for Bluetooth devices on the phone.

Success criteria: At least one of the Bluetooth motes is discovered.

Case 2

Actions: Check the number of created application it is possible to choose between.

Success criteria: It is possible to choose between a sender and receiver application.

Case 3

Actions: Start the "Sender" application on the phone.

Success criteria: The Sender main menu is displayed.

Case 4

Actions: Start the "Receiver" application on the phone.

Success criteria: The Receiver main menu is displayed.

Case 5

Actions:

1. Start the sender application on the phone.
2. Open the file types menu.
3. Create or load the file.
4. Search for Bluetooth devices.
5. Select a mote device.
6. Send the file.
7. Repeat for all file types.

Success criteria:

1. The menu allows the user to choose data type.
2. The file is created and saved in the directory c:\\sensor\\files.
3. The application starts a device search.
4. A list of devices in range is shown.
5. "Connection established" is shown on the mobile phone.
6. "Sending message" is shown on the mobile phone.

Case 6

Actions:

1. Start the receiver application on the phone.
2. Search for Bluetooth devices.
3. Select a mote device.
4. Send a message from the sender phone.

Success criteria:

1. The application starts a device search.
2. A list of devices in range is shown.
3. "Connection established" and "Listening for messages" is shown on the mobile phone.
4. The receiver starts receiving the message and shows the content when transmission is complete.

Case 7

Actions: Send a message from the sender phone.

Success criteria: The number of packets is shown on the display. This number corresponds to $\frac{1}{29}$ of the total message size.

Appendix B

nRF ShockBurst flow charts

This appendix contains flowcharts from the nRF905 datasheet[8] showing how the transceiver works in RX and TX mode.

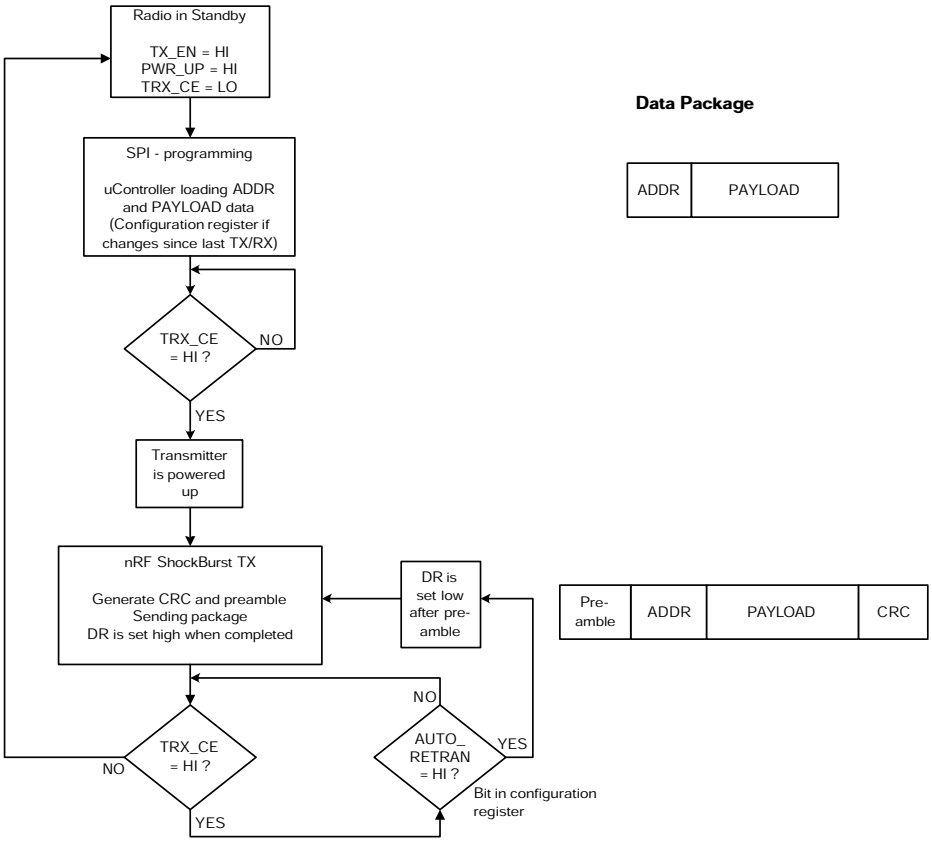


Figure B.1: *Flow chart showing the nRF905 ShockBurst transmit mode[8]*

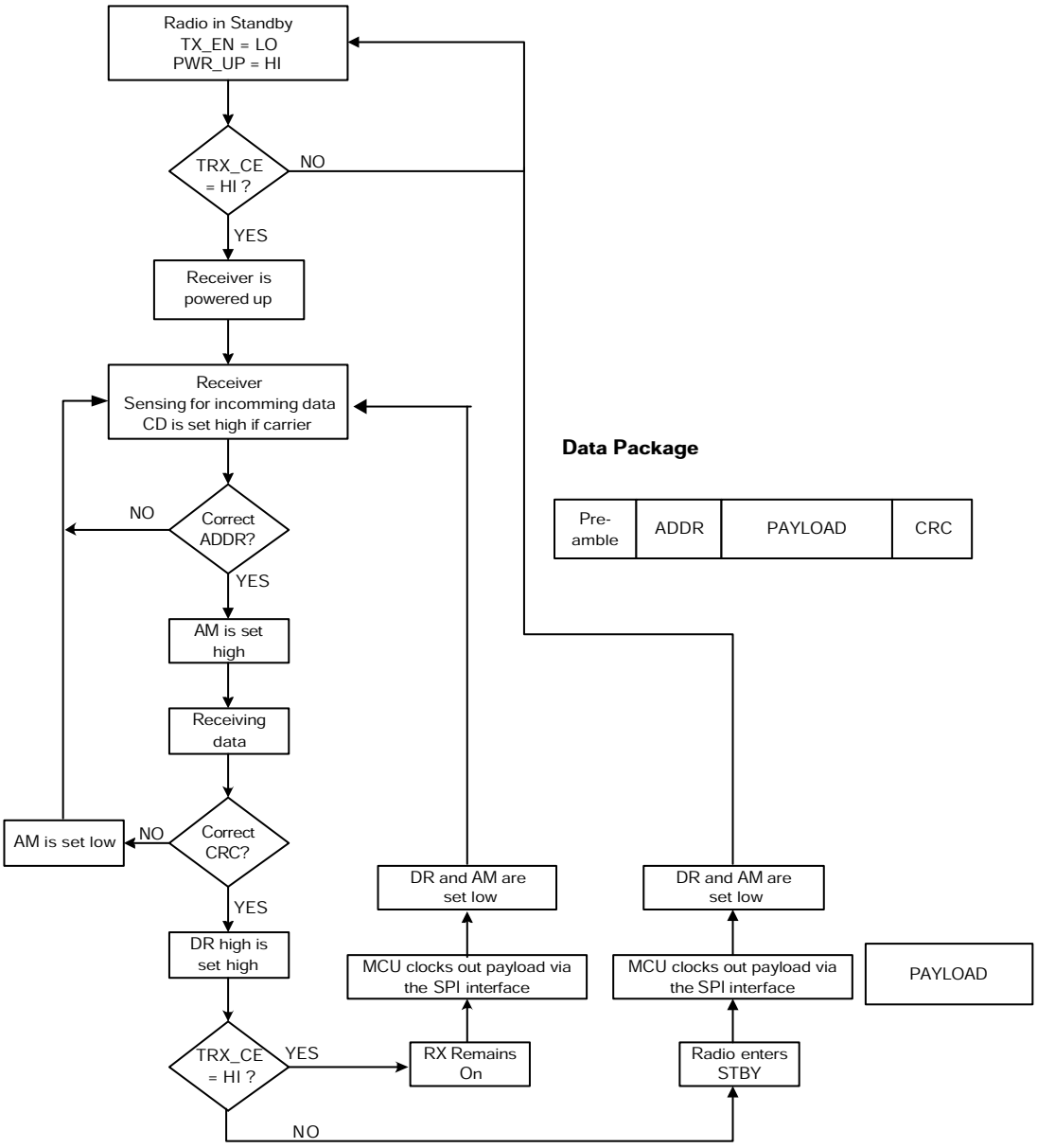


Figure B.2: *Flow chart showing the nRF905 ShockBurst receive mode[8]*

Appendix C

Bluetooth send/receive switching test

The mobile phones have to switch from receiving mode to sending mode or vice-versa many times. This happens because the size of a packet is only 29 bytes, therefore for quite big files a lot of packets will be created and for each packet sent/received a request/acknowledgement packet will be received/sent. It is necessary that the the system does not become slow because of the time to switch from a function to another one. In order to test if this speed is fast enough a little test program has been implemented in both mobile phones: A phone sends a packet and waits for a packet with the same size, the other phone does the opposite procedure, it waits for a packet and after receiving it sends another packet with the same size. This procedure is repeated continuously. The trials are done within different scenarios:

- Packet size: 1 byte or 29 bytes
- Loops: 10, 100 or 1000 loops

For each combination three measurements has been done, the results are shown in Figure C.1:

	1 Byte	29 Byte
10 Loop	0,28125	0,29687
	0,21875	0,28125
	0,28125	0,28125
100 Loop	1,90625	1,98437
	1,92187	1,96875
	1,93750	1,93750
1000 Loop	18,20312	18,03125
	18,14062	18,92187
	18,29687	18,04687

Figure C.1: *Test results of the switching in seconds*

The mean of each scenario is calculated and shown in Figure C.2.

	1 Byte	29 Byte
10 Loops	0,26041	0,28645
100 Loops	1,92187	1,96354
1000 Loops	18,21353	18,99999

Figure C.2: *The mean time of the switching in seconds*

The mean time spent for a loop, working out the values in Figure C.2, is 22 ms.

In Section 7.1.3 is shown that the time for sending one packet to a mote to another one is about 5 ms plus the delay to sense the carrier, in total is approximately 20 ms. This value is very close to the time calculated in this test for sending a packet from the mobile phone to a mote. The conclusion is that the mobile phone application creates an acceptable delay for the whole system.

Appendix D

Mobile phone application test

The test described in this section has been carried out to prove that the mobile phone application works as it is specified in the design. This has to be done before testing the entire system. In order to obtain this result a test application has been created to simulate the mote network. Figure D.1 shows the used testing scenario.

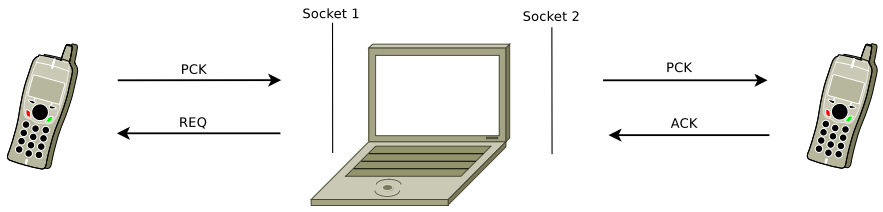


Figure D.1: *Mobile phones application test scenario.*

The applications running in the two mobile phones are the final applications. The test application running in the PC is implemented in Python 2.5, and it works as it is following explained :

- **It opens two sockets** one waiting for a connection from the first mobile phone, and one waiting from a connection from the second mobile phone.
- **It waits for packets from the first mobile phone**
- **For each packet received:**

  1. It forwards the packet to the second mobile phone through the second opened socket.
  2. It waits for an ACK packet from the second mobile phone.
  3. It sends a REQ packet to the first mobile phone.

The test program run in the PC, do not have to store the data in a buffer before it sends. Differently from how it is done in the mote network, each packet is requested only when the previous one has already been received from the receiver mobile phone, and an acknowledgement has been sent. However the mobile phone applications do not feel any differences, they work exactly as they would work with the mote network and therefore they are correctly tested.

Appendix E

Transmission range test

To find the optimal distance between the motes in the chosen network scenario and to avoid the hidden terminal problem as well limitting the number of collisions a range test of motes at different power settings has been performed.

The setup of the test was that the motes where in line of sight and the transmitting mote was sending a packet each second. The different transmission and receiving settings was as follows:

- Transmission power: -10 dBm, 2 dBm, 6 dBm or 10 dBm
- Receiver sensitivity: Normal sensitivity or reduced sensitivity

These settings resulted in 8 different tests, where the success criteria was that 10 packets where continuously received which is assumed to be a stable link optimal for the chosen network scenario. The test results can be seen in Table E.1

TX power	**High RX**	**Low RX**
-10	1 m	0.40 m
2	7.5 m	1 m
6	13 m	6 m
10	24 m	9 m

Table E.1: *The result of the transmission range test*

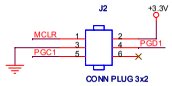
Appendix F

Motes hardware schematics

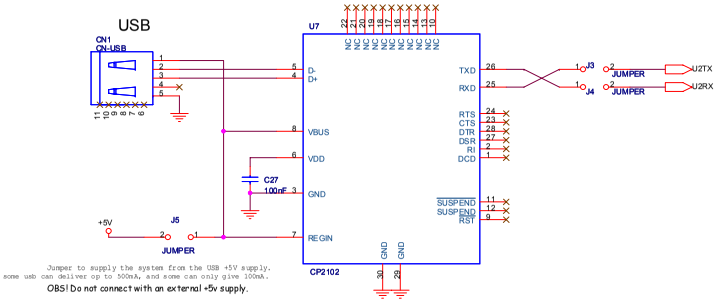
This appendix includes the hardware schematics of the mote used in the project. They are shown in the following pages.

MAIN SENSOR BOARD REV 2.0

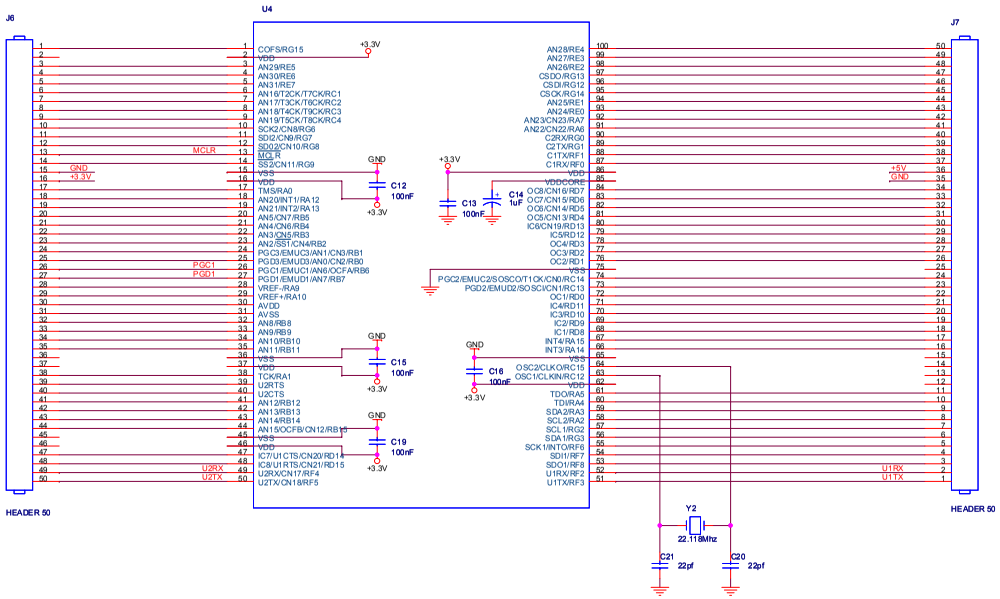
ICD2 DEBUGGER/PROGRAMMER



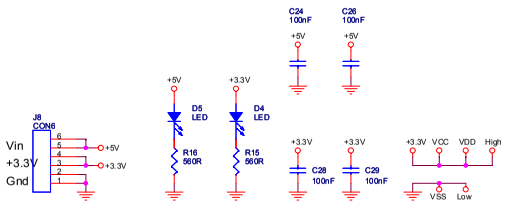
USB to RS232



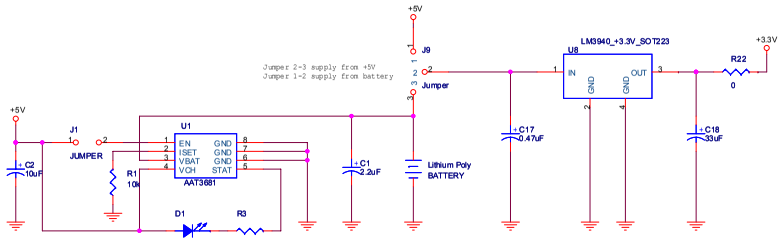
DSPIC33FJ256 DSP



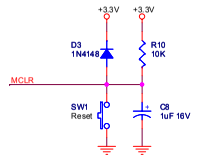
POWER LINES



SMART LITHIUM POLY BATTERY CHARGER MAX 300mA OUT +3.3V LOW DROP REGULATOR 0.5V, 1.0 A



MCLR RESET



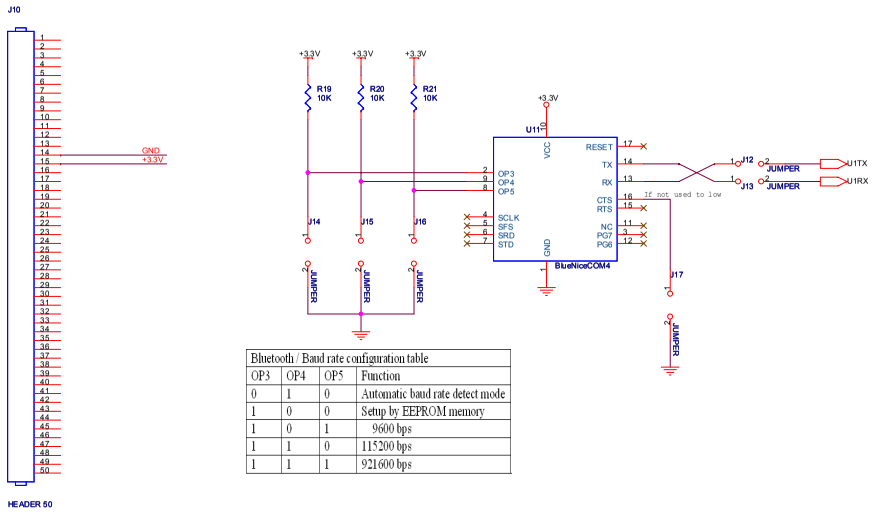
<Mobile Devices>
<AALBORG UNIVERSITY>
<OrgAddr1>
<OrgAddr2>

Title: MAIN SENSOR BOARD

Design file:	C:\DOCUMENTS AND SETTINGS\BIMMY\DOCUMENTS\ORCAD\DSPIC33F J256\PTI10\SENSORBOARD REV2\SENSOR BOARD2.DSN		
Date create:	Tuesday, January 30, 2007	Sheet:	2 of 2
Date modify:	Thursday, February 15, 2007	Size: A2	Rev: <2.0>

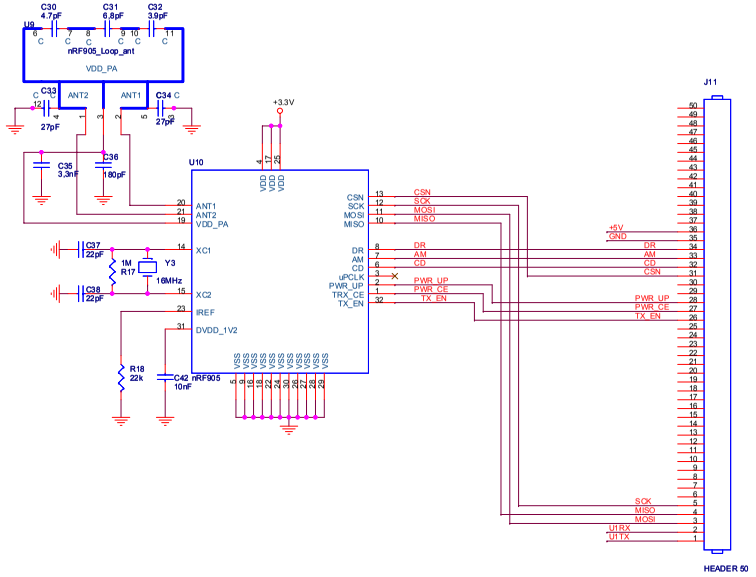
WIRELESS SENSOR BOARD REV 2.0

BLUETOOTH MODULE

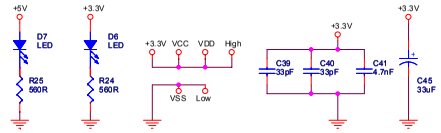


Bluetooth / Baud rate configuration table			
OP3	OP4	OP5	Function
0	1	0	Automatic baud rate detect mode
1	0	0	Setup by EEPROM memory
1	0	1	9600 bps
1	1	0	115200 bps
1	1	1	921600 bps

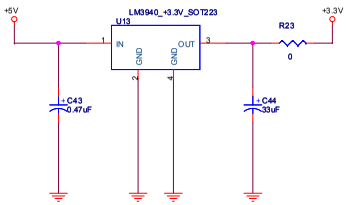
RF TRANSCEIVER 433Mhz



POWER LINES



+3.3V LOW DROP REGULATOR 0.5V, 1.0 A



<MobileDevices>
<OrgAddr2>
<OrgAddr3>
<OrgAddr4>

Title: <WIRELESS SENSOR BOARD>

Design file:	C:\DOCUMENTS AND SETTINGS\BIMMY\DOCUMENTS\ORCAD\DSPICS\F J2662\PT0\SENSORBOARD REV2\SENSOR BOARD2.DSN		
Date create:	Sunday, January 21, 2007	Sheet:	1 of 2
Date modify:	Thursday, February 15, 2007	Size: A2	Rev: <2.0>