IMPLEMENTATION OF THE IEEE 802.11a MAC LAYER IN C LANGUAGE

Bachelor thesis performed in Deparment of Electrical Engineering
by

Carlos Alonso Guillen

LiTH-ISY-EX-ET-0275-2004
2004-01-17

IMPLEMENTATION OF THE IEEE 802.11a MAC LAYER IN C LANGUAGE

Bachelor thesis in Deparment of Electrical Engineering for
Technical Engineering of Telecommunication in
Telecommunication Systems
Linköping Institute of Technology
by

Carlos Alonso Guillén

LiTH-ISY-EX-ET-0275-2004

Supervisor: Kent Palmkvist
Examinator: Kent Palmkvist
Linköping 2004-01-17

LINKÖPINGS UNIVERSITET

Avdelning, Institution
Division, Department

Institutionen för systemteknik
581 83 LINKÖPING

Datum
Date
2004-01-17

Språk
Language
Svenska/Swedish
X Engelska/English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats
Övrig rapport

ISBN

ISRN LITH-ISY-EX-ET-0275-2004

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2004/275/

Titel Implementering av IEEE 802.11a MAC-lagret i programspråket C
Title

Implementation of the IEEE 802.11a MAC layer in C language

Författare Carlos Alonso Guillen
Author

Sammanfattning
Abstract
Wireless communication is being developed in the last years day by day, there are several standards that talks about it. We are going to go through the IEEE standard 802.11 which talks about wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Looking this more carefully we will study MAC specifications and its environment.

The work that ISY department at Institute of Technology of Linkoping University has proposed is to design a MAC sublayer implementation for WLANs using C language programming and testing it with the test environment called "test bench". This test bench will simulate LLC sublayer and PHY layer, in this way, our MAC implementation will has to interact with it. Therefore we will simulate a wireless network where we are going to have a short number of stations and we are going to look at carefully the MAC sublayer response in an ad hoc network.

Nyckelord
Keyword
IEEE, 802.11, WLAN, MAC, C language

ABSTRACT

Wireless communication is being developed in the last years day by day, there are several standards that talks about it. We are going to go through the IEEE standard 802.11 which talks about wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Looking this more carefully we will study MAC specifications and its environment.

The work that ISY department at Institute of Technology of Linkoping University has proposed is to design a MAC sublayer implementation for WLANs using C language programming and testing it with the test environment called "test bench". This test bench will simulate LLC sublayer and PHY layer, in this way, our MAC implementation will has to interact with it. Therefore we will simulate a wireless network where we are going to have a short number of stations and we are going to look at carefully the MAC sublayer response. All the necessary steps for designing this MAC implementation are in IEEE standard 802.11.We are going to suppose an ad hoc network (it will not be necessary to use access point) using always the DCF access protocol and using collision avoidance medium sharing mechanism (CSMA/CA).

This MAC implementation design is going to have a flexible solution because of the C language is a very flexible programming language, we have tried to give the closest solution to the standard.

PREFACE

English
This thesis is the end of six hard years of work at Escuela Politécnica of Alcalá de Henares University and one semester at Linkoping University Institute of Technology, where I have tried to get the Technical Engineering of Telecommunication in Telecommunication Systems degree.

First of all, I would like to take the opportunity to thank the Socrates/Erasmus organization for let me study abroad with all kind of facilities, the work my both coordinators have done has been great.
I also would like to thank my family that has supported me during all these years at the university and Maria Jose for being always with me.

Of course, I have to thank Fernando that has being helping me with my studies always that I have needed it and also for being a great friend all this time.
All my classmates for all the good and bad times we have passed together and all my friends in general for listen to me and lean my always.
And finally I would like to thank Maria all the effort she has done and tell her it has been a pleasure for me to work with her.

Spanish
Este proyecto es el final de seis duros anos de trabajo en la Escuela Politécnica de la Universidad de Alcalá de Henares y de un semestre en el Instituto de Tecnología de la Universidad de Linkoping, siendo mi objetivo final obtener el titulo de Ingeniero Técnico de Telecomunicaciones especializado en Sistemas de Telecomunicación.

Antes de nada, me gustaría agradecer a la organización Socrates/Erasmus la oportunidad que me ha dado para estudiar fuera con todo tipo de facilidades y también al magnifico trabajo que mis dos coordinadores Antonio y Kent han hecho. Aprovecho también la oportunidad para agradecer a mi familia el haberme soportado durante estos años en la universidad y a María José por estar siempre a mi lado durante estos dos últimos años.

A Fernando por haberme ayudado con los estudios siempre que lo he necesitado y por haber sido un muy buen amigo durante tantos anos. A todos mis compañeros de clase por todos los buenos y no tan buenos momentos que hemos pasado juntos y por supuesto a todos los amigos en general que han estado siempre ahí escuchándome y apoyándome.
Y por ultimo me gustaría agradecerle a María todo el esfuerzo que ha hecho y decirle que ha sido un placer trabajar a su lado.

Carlos Alonso Guillén, Linkoping 2004-01-17.

CONTENTS

1. THIS REPORT

1.1. OVERVIEW

Chapter 2 and 3 is intended to give a general idea about the environment we are going to move in. These two chapters will give the necessary knowledge for understanding the idea of this work.
Chapter 4 describes the design decisions, in this chapter is shown how implementation was made.
Chapter 5 describes the design for MAC implementation
Chapter 6 shows the conclusions of this thesis.
The meaning of the abbreviations we have used can be seeing in appendix A.
Information about variables we have used in the programming code is shown in the appendix B.
Example of MAC implementation is shown in appendix C.
The Transmission part of MAC implementation is shown in appendix D.
Other files in relation with the MAC implementation are shown in appendix E.

1.2. PURPOSE

The main goal of this final year project is to develop a MAC implementation with the characteristics showed in the IEEE standard 802.11 (see chapter 4). For design it C language has been used and for simulating it a software called test bench (PUM10_) was provided by the ISY department at the Technologic Institute of Linkoping University.

1.3. PLANNING

To obtain the purpose work a time plan has been developed as follows:

- The first six weeks consisted of a study of the standard and several documents for trying to understand the standard, at first this was a problem because of the English language. At the same time I was studding another subject at Linkoping University during two months.
- Week 7 to 9 consisted of going through the diagrams and looking for the variables we were to use, it was difficult to identify the diagrams with the MAC implementation we wanted to design.
- Week 10 to 12 consisted of writing all the steps we had to follow in the C language program, studding C language and starting to write this document.
- Weeks 13 to 14 consisted of designing the MAC implementation and continue writing this document.
- Week 15 to 18 consisted of improving the MAC implementation and finishing some chapters of this document, for instance the conclusion, expected results and some appendix.
- Last week was for studying the presentation of this project and getting ready the format of this document.

Two persons have made this project and the work has been divided in two parts. For more information about this see Design Flow.

2. INTRODUCTION

This section gives a general idea for understanding the developed work. It only gives a little description about the field we are going to move in.

2.1. WIRELESS NETWORKS

Wireless networking is a means of providing ubiquitous computing. This means that it enables computing any time and any place. In a wireless network, all of the computers broadcast their information to one another using radio signals. We can find many advantages using a wireless network, some of them are these :

- The use by land, sea or air.
- Easy to install.
- Wireless LANs (for example) are cheaper than wiring the building.
- It comes in many forms and having different combinations of wire and wireless networking, this allow us to use it in many kinds of different situations (were telephone system has been destroyed for example).

In the another hand, it has some disadvantages, for example, capacity much slower than wired networks, error rates are more often and transmissions from different computers can interference with one another.
There are four types of wireless networks: WWANs, WLANs, WDANs and WPANs

2.2. WIRELESS LAN

A WLAN is a wireless local area network and enables traditional networking in buildings where installing cables may be difficult. Users do not require data jack connections or patch cables to connect to a wireless network. By eliminating these requirements, a WLAN facilitates ease of maintenance providing us with facilities that wired systems have not. A great benefit of WLAN structures are that they enable ad hoc networking, which means that, two or three devices can instantly participate in a network. With the adoption of the IEEE 802.11b standard, wireless equipment is easily integrated into existing network structures and can support data transfer speeds of 11 Mbps.

2.3. OSI

The international Standards Organization (ISO) has defined an architecture called the Open Systems Interconnection (OSI). The OSI reference model consists of seven protocol layers which are defined as follow :

1) Application : Provides different services to the applications
2) Presentation : Converts the information
3) Session : Handles problems which are not communication issues
4) Transport : Provides end to end communication control

5) Network : Routes the information in the network
6) Data Link : Provides error control between adjacent nodes
7) Physical : Connects the entity to the transmission media

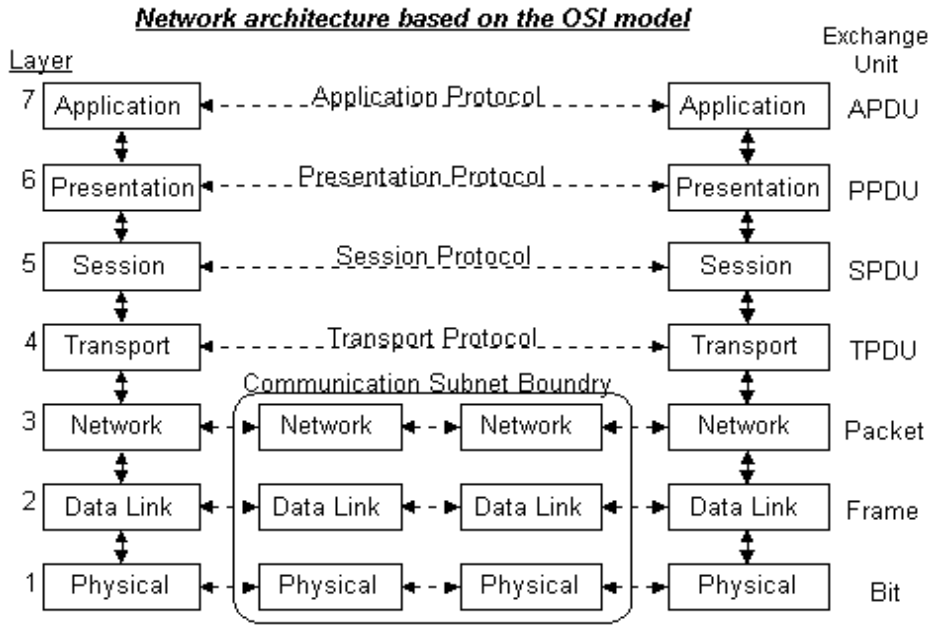

Figure 1. Network architecture based on the OSI model

The IEEE 802.11 standard, which is in focus in this report, is situated in the physical layer and data link layer. We can see in the figure 1 the data transformations being manipulated in each layer until finally it is sent in bits format in the physical layer.

2.4. THE DATA LINK LAYER

The main task of the data link layer is to take a raw transmission facility and transform it into a line that appears free of transmission errors in the network layer. It accomplishes this task by having the sender break the input data up into data frames (typically a few hundred bytes), transmit the frames sequentially, and process the acknowledgment frames sent back by the receiver. Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning of structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame.
The data link layer should provide error control between stations.
Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism must be employed in order to let the transmitter know how much buffer space the receiver has at the moment. Frequently, flow regulation and error handling are integrated. It has also to synchronize frames between stations.

The data link layer is divided into two sublayers: The Media Access Control (MAC) layer and the Logical Link Control (LLC) layer. The MAC sublayer controls how a computer on the network gains access to the data and permission to transmit it. The LLC layer controls frame synchronization, flow control and error checking.

2.5. THE IEEE 802.11 STANDARD

The IEEE 802.11 standard goes through WLAN Medium Access Control (MAC) and Physical Layer Specifications. It defines the protocol and compatible interconnection of data communication equipment via the "air", radio or infrared, in a local area network (LAN) using the carrier sense multiple access protocol with collision avoidance (CSMA/CA) medium sharing mechanism. The medium access control (MAC) supports operation under control of an access point as well as between independent stations. The protocol includes authentication, association, and reassociation services, an optional encryption/decryption procedure, power management to reduce power consumption in mobile stations, and a point coordination function for timebounded transfer of data. The standard includes the definition of the management information base (MIB) and specifies the MAC protocol in a formal way, using the Specification and Description Language (SDL).
The radio implementations of the PHY specify either a frequency-hopping spread spectrum (FHSS) supporting 1 Mbit/s and an optional 2 Mbit/s data rate or a direct sequence spread spectrum (DSSS) supporting both 1 and 2 Mbit/s data rates.
The area covered by this standard is undergoing evolution.
The architectural view divide the system in two parts : the MAC sublayer and PHY layer which correspond to the lowest layers of de ISO/IEC basic reference model of Open Systems Interconnection (OSI).

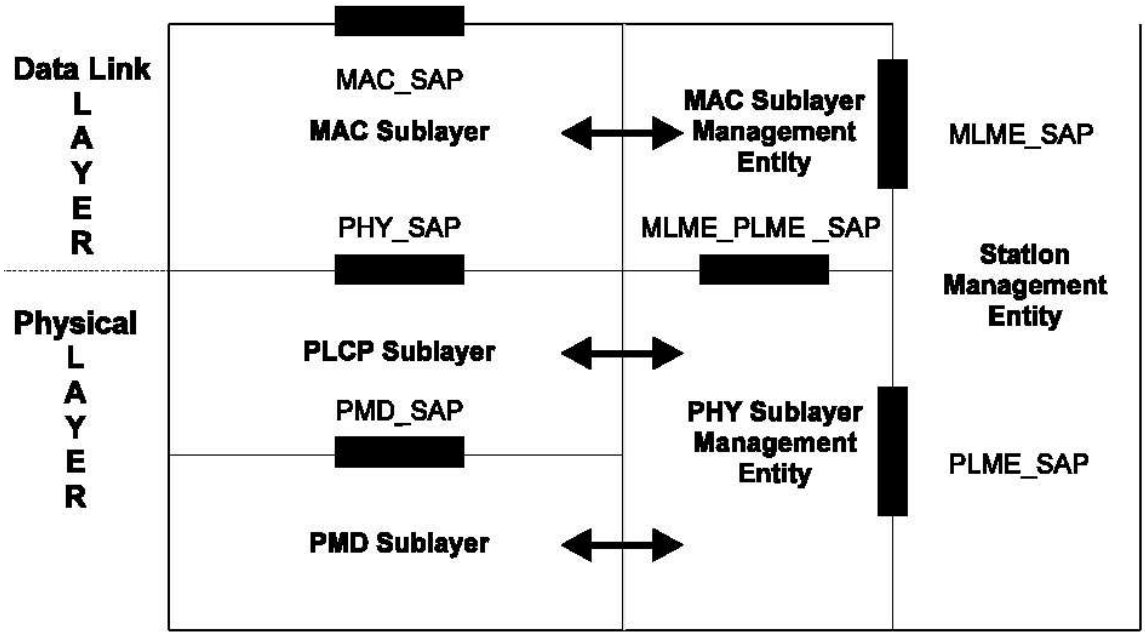

Figure 2. Portion of the ISO/IEC basic reference model covered in 802.11 standard

2.6. THE MAC SUBLAYER

This chapter is a short introduction to the MAC sublayer very useful for understanding the way MAC works and which is pointing to the aim of this project.

2.6.1. KINDS OF NETWORKS

There are two ways of implement a WLAN network, one of them is the ad hoc network where stations can connect to each other using an authentication protocol and accessing to the medium with a distributed coordination function (DFC). This kind of network will not need to be supported by an access point (AP) and it is denoted an IBSS (independent basic service set) in the standard.
Any collection of stations that communicates to each other inside the same wireless surface is called BSS (basic service set). The ESS (extended service set) is a set of one or more interconnected BSSs.
The another kind of network is denoted infrastructure network_ and it will use the access point to communicate the stations, therefore the communication between stations will be in two steps. This kind of implementation also use DFC for accessing to the medium.

2.6.2. MAC ARCHITECTURE

The way stations access to the medium can be achieved by two different methods :

Distributed coordination function (DCF):
It is the fundamental access method of IEEE 802.11 MAC and it is based in a carrier sense multiple access with collision avoidance (CSMA/CA). It will be implemented in all the STAs. When a STA wants to transmit it shall sense the medium to determine if another STA is transmitting and it must ensure the medium is idle for trasmitting. If the medium is busy it will wait before beginning the transmission. After deferral, or prior to attempting to transmit again immediately after a successful transmission, the STA shall select a random backoff interval and shall decrement the backoff interval counter while the medium is idle. Under various circumstances to further minimize collisions STAs can use exchange short control frames RTS/CTS after determining that the medium is idle and after any deferrals or backoffs, prior to data transmission The CSMA/CA distributed algorithm mandates that a gap of a minimum specified duration exist between contiguous frame sequences.

Point coordination function (PCF):
This method is only usable on infrastructure network configurations. It uses a point coordinator (PC), which shall operate at the access point of the BSS, to determine which STA currently has the right to transmit so as to eliminate contention for a period of time. The PCF uses a virtual carrier-sense mechanism aided by an access priority mechanism. PCF traffic shall have priority access to the medium over STAs in overlapping BSSs operating under the DCF access method.

The DCF and the PCF shall coexist in a manner that permits both to operate concurrently within the same BSS. When a PC is operating in a BSS, the two access methods alternate, with a contention-free period (CFP) followed by a contention period (CP).

2.6.3. DISTRIBUITED COORDINATION FUNCTION (DCF)

DCF is the medium access protocol used for accessing to the medium through the use of CSMA/CA and a random backoff time following a busy medium condition. In addition, all directed traffic uses immediate positive acknowledgment (ACK frame) where retransmission is scheduled by the sender if no ACK is received.
The CSMA/CA protocol is designed to reduce the collision probability between multiple STAs accessing a medium. When there are a several STAs waiting for transmitting data a random backoff procedure to resolve medium contention conflicts is used.
Carrier sense shall be performed both through physical and virtual mechanisms to determinate the state of the medium (idle/busy).

The virtual carrier-sense mechanism is achieved by distributing reservation information announcing the impending use of the medium. It exchanges RTS and CTS frames before the data frame to reserve the medium and to announce all STAs the period of time that the medium is to be reserved. This mechanism is referred to as the network allocation vector (NAV), this is a counter which is decreased until zero and when zero comes the medium is idle. The NAV maintains a prediction of future traffic on the medium based on duration information that is announced in RTS/CTS frames and MAC headers

The physical carrier-sense mechanism shall be provided by the PHY.

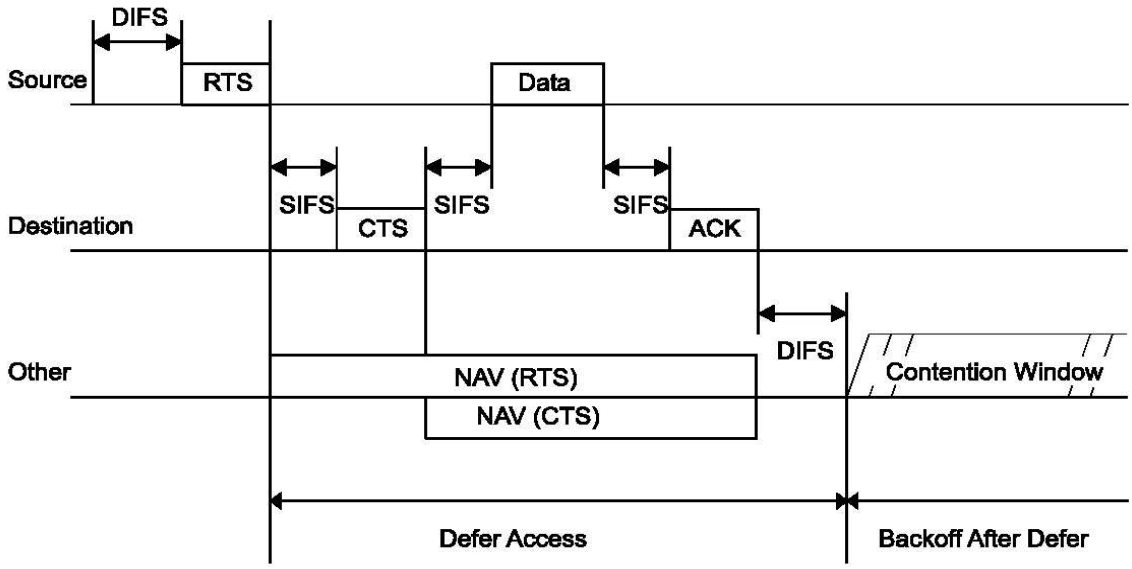

Figure 3. RTS/CTS, data, ACK and NAV setting in DCF

2.6.4. MAC FRAMES

MAC frames are made up with these three basic fields:

• MAC header, where these fields are : frame control, duration, address and frame sequence control.

Frame control subfields : Protocol Version, Type, Subtype, To DS, From DS, More Fragments, Retry, Power Management, More Data, Wired Equivalent Privacy (WEP), and Order.

- Duration/ID field : it is 16 bits in length. It contains the frame duration value.
- Address fields : there are four address fields in the MAC frame format. These fields are used to indicate source address, destination address, transmitting station address, and receiving station address.
- Sequence Control field : it is 16 bits in length and consists of two subfields, the Sequence Number and the Fragment Number.

• The Frame Body is a variable length field that contains information specific to individual frame types and subtypes
• The FCS field is a 32-bit field containing a 32-bit CRC.

There are three kinds of MAC frames:
• Data frames.
• Control frames. For example ACK frames or RTS/CTS frames.
• Management frames. For example Beacon frames or TIM frames.
The general MAC frame format has this aspect :

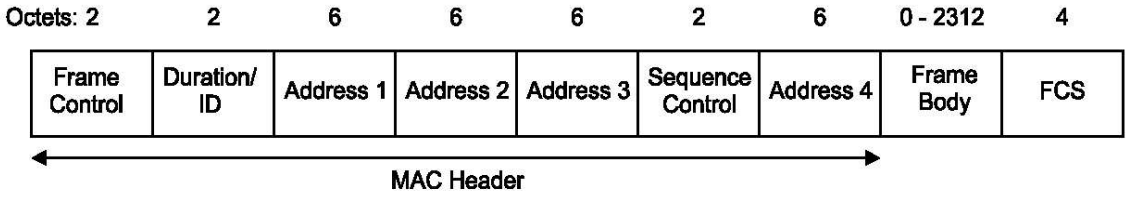

Figure 4. MAC frame format

2.6.5. ACKNOWLEDMENTS

The reception of some frames, requires the receiving STA to respond with an acknowledgment (ACK frame), if the FCS of the received frame is correct
Lack of reception of an expected ACK frame indicates to the source STA that an error has occurred. The acknowledge (ACK) frame is 14 bytes in length.

It has four fields to considerate:

- Frame control field
- Duration/ID field time to transmit the subsequent data or management frame, an ACK frame, and two SIFS intervals.
- The address field (RA) identifies the individual MAC address of the station to ACK is sent. In the ACK frame, the RA is always an individual address. The RA value is taken directly from the address 2 field of the immediately preceding data.
- Frame check sequence field (FCS)

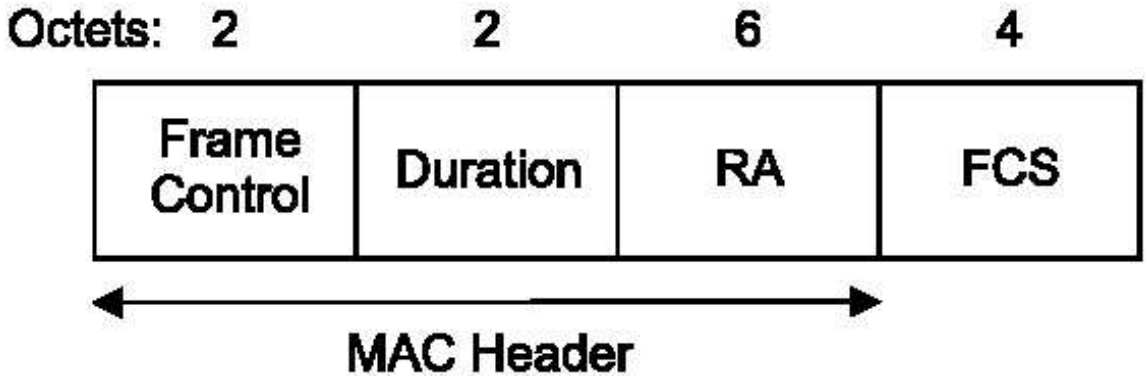

Figure 5. Acknowledge (ACK) Frame

2.6.6. INTERFRAME SPACE (IFS)

The time interval between frames is called the IFS. Four different IFSs are defined to provide priority levels for access to the wireless media; they are listed in order, from the shortest to the longest.

- SIFS, short interframe space : it is used for an ACK frame, a CTS frame, the second or subsequent MPDU of a fragment burst, and by a STA responding to any polling by the PCF.

- PIFS, PCF interframe space : it is used only by STAs operating under the PCF to gain priority access to the medium at the start of the CFP.

- DIFS, DCF interframe space : it is used by STAs operating under the DCF to transmit data frames (MPDUs) and management frames (MMPDUs).

- EIFS, extended interframe space : it is used by the DCF whenever the PHY has indicated to the MAC that a frame transmission was begun that did not result in the correct reception of a complete MAC frame with a correct FCS value.

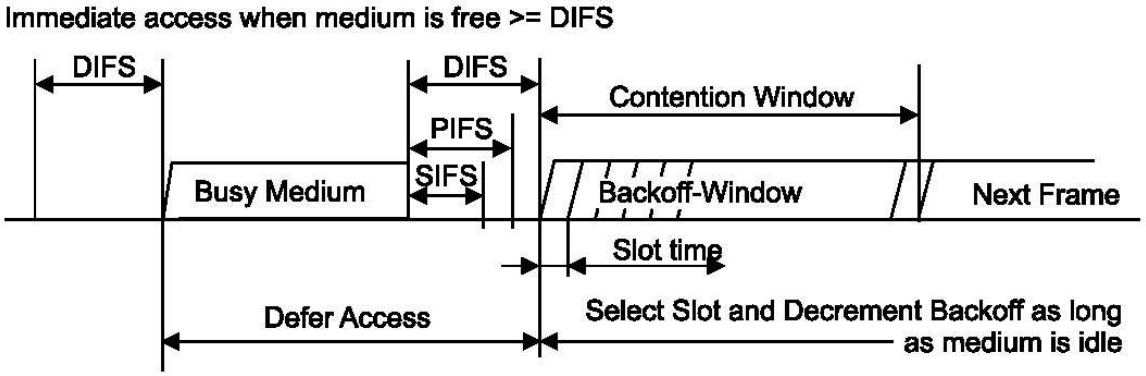

Figure 6. IFS relationships

2.6.7. BASIC ACCESS

A STA may transmit a pending MPDU when it is operating under the DCF access method, either in the absence of a PC, or in the CP of the PCF access method, when the STA determines that the medium is idle for greater than or equal to a DIFS period.
If, under these conditions, the medium is determined by the carrier-sense mechanism to be busy when a STA desires to initiate the initial frame of the frame, the random backoff algorithm shall be followed.
If, when transmitting or retransmitting an MPDU, there is not enough time remaining in the dwell to allow transmission of the MPDU plus the acknowledgment (if required), the STA shall defer the transmission by selecting a random backoff time, using the contention window (CW).

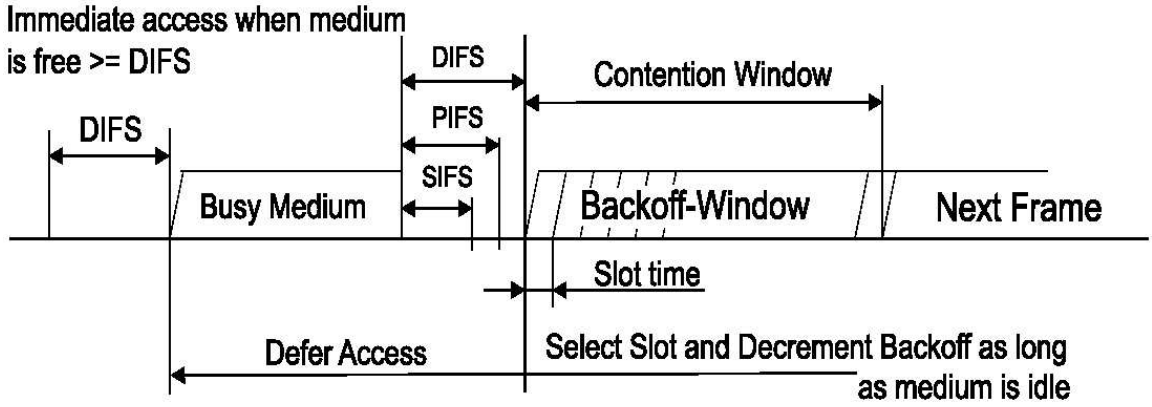

Figure 7. Basic access method

2.6.8. FRAGMENTATION AND DEFRAGMENTATION

MAC services data unit (MSDU) or MAC management protocol data unit (MMPDU) needs to be fragmented into smaller MAC level frames call MPDUs when the length of the frame is greater than a Fragmentation Threshold. This increase the probability of successful transmission of the MSDUs or MMPDUs. The process of recombining MPDUs into a single MSDU or MMPDU is defined as defragmentation.

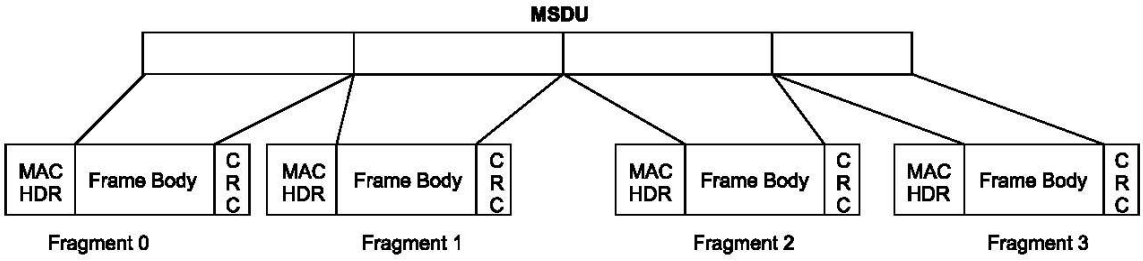

Figure 8. Frame fragmentation

2.6.9. CONTROL OF THE CHANNEL

The SIFS is used to provide an efficient MSDU delivery mechanism. Once the STA has contended for the channel, that STA shall continue to send fragments until either all fragments of a single MSDU or MMPDU have been sent, an acknowledgment is not received, or the STA is restricted from sending any additional fragments due to a dwell time boundary. When the source STA transmits a fragment, it shall release the channel, then immediately monitor the channel for an acknowledgment. When the destination STA has finished sending the acknowledgment, the SIFS following the acknowledgment shall be reserved for the source STA to continue (if necessary) with another fragment. The STA sending the acknowledgment shall not transmit on the channel immediately following the acknowledgment. If the source STA does not receive an acknowledgment frame, it shall attempt to retransmit the failed MPDU, after performing the backoff procedure and the contention process. The destination STA shall receive the fragments in order (since the source sends them in order, and they are individually acknowledged).

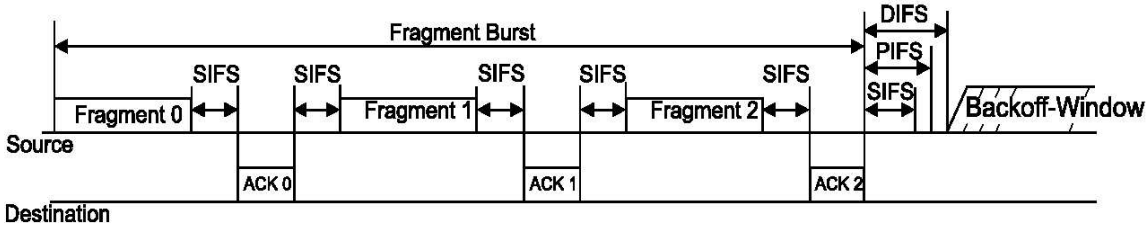

Figure 9. Transmission of a multiple-fragment MSDU using SIFS

3. BACKGROUND

This section is a description of the background used to design the MAC layer implementation, it explains what kind of tools have been used and the way they have been used.

3.1. TEST BENCH

The test bench is the environment where we are going to test and evaluate our MAC layer implementation, it will simulate a wireless network with a number of stations and channels. In this way, we are going to design the MAC sublayer implementation and the test bench will take charge of simulate LLC sublayer and PHY layer.
The test bench system is going to be divided in four main parts :

- Application: it obtains the information required about stations, channels and configuration files for using it during the test.

- MAC implementation: this is what we want to test, it will be the nucleus of the this thesis.

- Configuration file: it will gives us information about the structure of the simulated network.

- Log output file: this is the file where we are going to see the results of the test, for making this possible, we have to call log file in the MAC implementation each time we want to communicate anything to the output.

A couple of the main advantages of this software are described :

- It is really easy to use it, it exists a user manual with all the information necessary (PUM10_).
- It provides the facilities for simulating differents MAC sublayer implementations besides we are not going to need to built all the system.

The only thing we need to know is how is going to be the interface between the test bench and our MAC implementation. All the steps for configurating our simulated network are given in the user manual.

3.1.1. THE APPLICATION

Like we have said before the application obtains the information required about stations, channels and configuration files for using it during the test. Once we have ran the test bench it exists a number of available user commands to give the necessary facilities for moving within the test bench.

Command	Description
add *station_id filename*	Add test vector to station station_id
load *filename*	Load configuration file
restart	Restart the test bench
run *time time_units*	Run test
step	Run until next event
quit	Exit the test bench

Figure 10. User commands

3.1.2. CONFIGURATION FILE

It gives information about the structure of the simulated network.
This file is divided in three blocks :

- General block: it gives information about the network we are going to simulate and the administrative information about the test.

Identifier	Description
nr_of_stations	The total number of stations in the network.
log_file	Name of the log file. Stated within " ".
log_level	Log level. The levels are none, low, medium, high and debug. Stated within " ".

- Station block: we are going to have one station block for each station in the network and this block will define the station in the simulated network.

Identifier	Description
id	Station id. Unique positive integer.
name	Name of the station. Stated within " ".
test_vector_file	Test vector file for the station. Stated within " ".
mac_implementation	Mac implementation file for the station. Stated within " ".

- Channels block: it defines the channels in the simulated network.

Identifier	Description
station A channel [B, "type"]	Station A has a outgoing channel to station B of type "type".

Each block start with the block name in < >, for example <general>. The order of the blocks is important. If the blocks are stated in wrong order the application may not be able to recognize all blocks. Correct order: General first, station second and the channels block last. All text outside the blocks (< >) are ignored by the application.

3.1.3. TEST VECTOR

The test vector is the file that is going to take charge of simulate LLC sublayer. It will be sent from the source MAC to the destination MAC at the especified time with a particular information. These are the identifiers which build the test vector :

Identifier	Description
TIME	Time when test vector will be sent. Must be given in nano seconds smaller than 2^{63}.
MACADDR (first)	Source address of MAC.
MACADDR (second)	Destination address of MAC.
DATA	Data package. Must be less than 4608 hex digits.
PRIO	Priority. 0=Contention, 1=ContentionFree. See [802.11] chapter 6 for more information.
SCLASS	Service class. 0=ReorderableMulticast, 1=StrictlyOrdered. See [802.11] chapter 6 for more information.

Figure 11. Test Vector. Identifiers

3.2. MAC IMPLEMENTATION

In our MAC implementation we need to know how to communicate MAC and test bench to each other, for this we are going to have some available functions to interact with the test bench and we are going to design some required functions the test bench is going to use to interact with our MAC sublayer. We are going to simulate a MAC implementation using RTS, CTS, DATA and ACK frames while connected to a BSS and using the DCF access protocol. All the stations simulated in the BSS will have to access to the medium using CSMA/CA protocol. The communication will be an ad hoc communication without using access point (AP) in the BSS.

3.2.1. AVAILABLE FUNCTIONS

These are the functions we are going to use to interact with the test bench. They are all called by using the function pointers provided in the MAC_SETUP structure (see "MAC_SETUP"). The first argument (staid) of all functions is the id of the Station object the MAC implementation is associated with. This is also contained in MAC_SETUP. We are going to describe now these functions and the purpose wrote in the 802.11 standard.

void MA_UNITDATA_indicate(int staid, MA_UNITDATA)
This function defines the transfer of an MSDU from the MAC sublayer entity to the LLC sublayer entity. The MA_UNITDATA.indication function is passed from the MAC sublayer entity to the LLC sublayer entity or entities to indicate the arrival of a frame at the local MAC sublayer entity. Frames are reported only if they are validly formatted at the MAC sublayer, received without error, received with valid (or null) WEP encryption, and their destination address designates the local MAC sublayer entity.

void MA_UNITDATA_STATUS_indicate(int staid, MA_UNITDATA_STATUS)
This function has local significance and provides the LLC sublayer with status information for the corresponding preceding MA-UNITDATA.request function. The MA-UNITDATA-STATUS.indication function is passed from the MAC sublayer entity to the LLC sublayer entity to indicate the status of the service provided for the corresponding MA-UNITDATA.request function.

void PHY_DATA_request(int staid, octet_t data)
This function defines the transfer of an octet of data from the MAC sublayer to the local PHY entity. It is generated by the MAC sublayer to transfer an octet of data to the PHY entity. This function can only be issued following a transmit initialization response (PHY-TXSTART.confirm) from the PHY layer.

void PHY_TXSTART_request(int staid, TXVECTOR vect)
This function is a request by the MAC sublayer to the local PHY entity to start the transmission of an MPDU. It will be issued by the MAC sublayer to the PHY entity whenever the MAC sublayer needs to begin the transmission of an MPDU.

void PHY_TXEND_request(int staid)
This function is a request by the MAC sublayer to the local PHY entity that the current transmission of the MPDU be completed. It will be generated whenever the MAC sublayer has received the last PHY-DATA.confirm from the local PHY entity for the MPDU currently being transferred.

void PHY_CCARESET_request(int staid)
This function is a request by the MAC sublayer to the local PHY entity to reset the clear channel assessment (CCA) state machine. It is generated by the MAC sublayer for the local PHY entity at the end of a NAV timer. This request can be used by some PHY implementations that may synchronize antenna diversity with slot timings.

void registerTimeOut(int, time_t, void (*)(int), int)
This function is used to register a time-out with the test bench. When the specified time has elapsed, the function pointer is called with the integer argument provided.

void log(char* message, log_level_t)
Writes a message (a null terminated string) to the log file.

3.2.2. REQUIRED FUNCTIONS

These are the functions we have implemented in our MAC implementation and the functions used by the test bench during the simulation.
We are going to describe now these functions and the purpose wrote in the 802.11 standard.

void MA_UNITDATA_request(MA_UNITDATA data)
This function requests a transfer of an MSDU from a local LLC sublayer entity to a single peer LLC sublayer entity. It is generated by the LLC sublayer entity whenever an MSDU is to be transferred to a peer LLC sublayer entity.

void PHY_DATA_confirm()
This function is issued by the PHY sublayer to the local MAC entity to confirm the transfer of data from the MAC entity to the PHY sublayer. It will be issued by the PHY sublayer to the MAC entity whenever the PLCP has completed the transfer of data from the MAC entity to the PHY sublayer. The PHY sublayer will issue this primitive in response to every PHY-DATA.request primitive issued by the MAC sublayer.

void PHY_TXSTART_confirm()
This function is issued by the PHY sublayer to the local MAC entity to confirm the start of a transmission. The PHY sublayer will issue it in response to every PHY-TXSTART.request function issued by the MAC sublayer. This function will be issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHY-TXSTART.request from the MAC entity and is ready to begin receiving data octets.

void PHY_TXEND_confirm()
This function is issued by the PHY sublayer to the local MAC entity to confirm the completion of a transmission. The PHY sublayer issues it in response to every PHY-TXEND.request function issued by the MAC sublayer. It will be issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHY-TXEND.request immediately after transmitting the end of the last bit of the last data octet indicating that the last data octet has been transferred.

void PHY_CCARESET_confirm()
This function is issued by the PHY sublayer to the local MAC entity to confirm that the PHY has reset the CCA state machine. It is issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHYCCARESET.request.

void PHY_DATA_indicate(octet_t)
This function indicates the transfer of data from the PHY sublayer to the local MAC entity. The PHY-DATA.indication is generated by a receiving PHY entity to transfer the received octet of data to the local MAC entity.

void PHY_CCA_indicate(CCASTATE)
This function is an indication by the PHY sublayer to the local MAC entity of the current state of the medium. It is generated every time the status of the channel changes from channel idle to channel busy or from channel busy to channel idle. This includes the period of time when the PHY sublayer is receiving data.

void PHY_RXSTART_indicate(RXVECTOR)
This function is an indication by the PHY sublayer to the local MAC entity that the PLCP has received a valid start frame delimiter (SFD) and PLCP Header. It is generated by the local PHY entity to the MAC sublayer whenever the PHY has successfully validated the PLCP Header error check CRC at the start of a new PLCP PDU. It will indicate MAC entity the arrival of the new frame.

void PHY_RXEND_indicate(RXERROR)
This function is an indication by the PHY sublayer to the local MAC entity that the MPDU currently being received is complete. It is generated by the PHY sublayer for the local MAC entity to indicate that the receive state machine has completed a reception with or without errors.

void initialise(MAC_SETUP)
This function will setup the MAC for test. If called during test it shall reset the MAC to the initial state.

void destroy()
This function is called at the end of a test. It will clean up after test by deallocating used memory, closing any opened files etc.

3.2.3. USED DATA TYPES

This section describes the special data types used in the interface functions and some component data types they use.

macaddr_t
Type used for MAC addresses. MAC addresses are 48-bits (6 byte). Defined as octet_t[6].

prio_t
enum prio_t = {contention, contention_free};

sclass_t
enum sclass_t = {reorderable_multicast, strictly_ordered};

tstat_t
enum tstat_t = {success, undeliv_unack, excess_len,
non_null_src_rout, prio_not_supp, sclass_not_supp, prio_not_av,
sclass_not_av, undeliv_time, undeliv_no_bss,
undeliv_null_encrypt};

MAC_SETUP
Struct used to set up a MAC implementation before test. Contains pointers to the available functions as well as the station id.

```
struct MAC_SETUP
{
int staid;
void (*MA_UNITDATA_indicate)(int, MA_UNITDATA)
void (*MA_UNITDATA_STATUS_indicate)(int,
MA_UNITDATA_STATUS);
void (*PHY_DATA_request)(int, octet_t)
void (*PHY_TXSTART_request)(int, TXVECTOR)
void (*PHY_TXEND_request)(int)
void (*PHY_CCARESET_request)(int)
void (*registerTimeOut)(int, time_t, void (*)(int), int)
void (*log)(char*, log_level_t)
}
```

MA_UNITDATA
Struct used for sending data between the LLC layer and MAC layer.

```
struct MA_UNITDATA
{
macaddr_t saddr;
macaddr_t daddr;
rinfo_t rinfo = NULL;
octet_t* data;
int datalen;
rstat_t rstat = SUCCESS;
prio_t prio;
sclass_t sclass;
}
```

MA_UNITDATA_STATUS
struct MA_UNITDATA_STATUS
{
macaddr_t saddr;
macaddr_t daddr;
tstat_t tstat;
prio_t prov_prio;
sclass_t prov_sclass;
}

TXVECTOR
struct TXVECTOR
{
int rate = 54;
int length;
}

CCASTATE
enum CCASTATE = {BUSY, IDLE};

RXVECTOR
struct RXVECTOR
{
int rate = 54;
int length;
}

RXERROR
enum RXERROR = {no_error, format_violation, carrier_lost,
unsupported_rate};

3.3. THE C LANGUAGE

C language is located in an intermediate level between Pascal and Assembler. It tries to be a high level language with the low's level versatility. It was designed at the same time as the UNIX operating system and it is mainly oriented for working in this environment. C's compilers are today powerful and easy to design, they permit to increase the programmer's productivity. C is today the language of choice for many programmers, because of its great flexibility and wide applicability. Its best features are the powerful operator set, the terseness (less typing) and its large standard library.

3.3.1. WHY C ?

Is known that the nucleus of this thesis is the MAC sublayer implementation, this one is going to be designed in C language, the reason is simple, this is the language we have to use for interacting with the test bench. This programming language gives us the opportunity and facilities for creating a powerful MAC implementation according to the steps written in 802.11 standard.

4. DESIGN FLOW

The design of the MAC implementation has been divided in two parts, the transmission and the reception. The reason of this division is because two persons have designed this MAC implementation and the work has had to be divided. In this way, the aim of this document is going to be the transmission while the another document will point to the reception part in the MAC implementation. It is known that for the understanding of one part we need to know how the another one works, this is the reason because now we are going to describe the steps we have to follow for all the design.
What functions are going to be developed in the transmission part and what in the reception part? This question is solved as follows:

Transmission: void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data);
int fduration (Tframetype frametype, int lenght);
BOOL backoff (MAC *mac,unsigned int a, unsigned int b, int c);
int GetRandomNumber(int iMaximum);
Void PHY_DATA_confirm(MAC* mac);
Void PHY_TXSTART_confirm(MAC* mac);
Void PHY_TXEND_confirm(MAC* mac);

Reception: void PHY_DATA_indicate(MAC* mac, octet_t data);
Void PHY_CCA_indicate(MAC* mac, CCASTATE state);
Void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv);
Void PHY_RXEND_indicate(MAC* mac, RXERROR rxe);
Void PHY_CCARESET_confirm(MAC* mac);

Void initialise_MAC(MAC* mac, MAC_SETUP* macs);
Void destroy_MAC(MAC* mac);
MAC* create_MAC(void);
void wait(int iTime);
void FillSequence(MAC *mac);
BOOL IsMediumBusy(MAC *mac);
BOOL TestChannelIdleAndBackOff(MAC *mac,int iTimer);
BOOL TestChannelIdleAndBackOffNoWait(MAC *mac);
void RTSTimeout(MAC *mac,int iValue);
void ACKTimeout(MAC *mac,int iValue);
void TimeoutIdle(MAC *mac,int iValue);
void test (MAC* mac, int arg)

For more information about reception part see Appendix D.

4.1. STEPS TO FOLLOW IN THE DESING

The PUM is the test bench where we are going to introduce our MAC implementation, it is going to be the responsible for transmitting and receiving the frames in the system. The PUM will take charge of PHY layer and LLC sublayer, for this reason it is going to have some available functions that our MAC implementation is going to use. In the another hand we are going to develop some required functions that PUM is going to call to interact with this MAC layer.
To develop this idea we are going to design the MAC sublayer of one station and after this we will simulate the test environment for a number of stations using PUM software. We will divide the problem in two steps:

- Writing MAC implementation.
- Simulate this MAC implementation establishing the test configuration (configuration file) for N stations.

Finally, our MAC layer is going to be a set of several functions that are going to describe the behavior shown in the IEEE standard 802.11. Now we are going to define when are we going to use available function from our MAC implementation and the steps we have followed for designing required functions in C language.

4.2. WHEN ARE WE GOING TO USE AVAILABLE FUNCTIONS?

These functions are written in the PUM but we are going to call them from our MAC implementation. Now we give a short description about how we have used them.

4.2.1. Void MA_UNITDATA_indicate (int staid, MA_UNITDATA)

MAC will use it to give data to LLC (PUM in our case), with this function MAC entity is going to tell LLC that has received a frame. The end of the reception of one frame will be one call to this function. One call to this function from wherever our program is, it means the end of reception.

4.2.2. Void MA_UNITDATA_STATUS_indicate (int staid, MA_UNITDATA_STATUS)

With this function our MAC implementation is going to tell LLC (PUM) the status and situation of one transmission it has ordered. We are going to call this function each time we send a frame and without looking at the state of the transmission (success/fail).

4.2.3. PHY_DATA _request

MAC will call this function each time it wants to transmit one byte (octet). We are going to call this function once for byte. We will implement the access to the medium before to call this function.

Once the STA has got the access to the medium we will follow these steps:

- We will call PHY_TXSTART_request (I want to begin transmission).
- We will call several times PHY_DATA_request once for each byte of the transmitting MPDU.
- When the MPDU has been transmitted we will call PHY_TXEND_request.

4.2.4. PHY_CCARESET_request

MAC will call this function, for instance, when its NAV set to 0, this is to inform PHY layer that it can continue communicating the changes in its state.
The logic control will defuse the physical control during a time (when the STA cannot transmit) and when this time expires, the STA will use this function for advising PHY layer that it want to know what is happening in the medium. At this moment the physical control will determinate the channel occupation.

4.2.5. Log

This function is not coming from IEEE, it is coming from PUM and it is going to be the only way to know the simulation results. In the required functions we will call this function each time something is going to change. When the simulation finish we can look at this log file and see the steps our MAC implementation has followed.

4.2.6. RegisterTimeOut

This function is used to register a time-out with the test bench, it must to be implemented in our MAC implementation.

4.3. REQUIRED FUNCTIONS

4.3.1. MA_UNITDATA_request (MA_UNITDATA data)

PUM will call this function taking charge of LLC for sending a data frame. In the body of this function we are going to initiate all the necessary actions for transmitting one data frame.
With the call to this function we will receive a type MA_UNITDATA variable where addresses, data, length, priority and service class are defined.

Steps in the body of this function:

- We check the actual STA id (staid) and keep it.
- We start initializing some fields and variables, and computing some header fields (for instance, duration).
- A pointer must point to the beginning of the header and the data body. We have to take care about the length of the header (is going to be a constant value in each different kind of frame) for this case its value is 30 bytes. The length of the body is a parameter (data length).
- Is the source address the same as the address of the STA we are now? If it is not we have to advise log that an address error has happened. If it is the same the execution will continue.
- Is the data length longer than the maximum? If it is we have to advise log and if it is not the execution will continue.
- Is the data length lower than the minimum? If it is we have to advise log and if it is not the execution will continue.
- Set addresses field of the frame with the address we have received.
- Set sequence control number of the frame (it is another field of the header).
- When we have filled all the header fields we have to inform log file about the length of the data we have received from LLC.
- Is the frame length longer than RTSThreshold? If It is we will send RTS before the data frame, if it is not it will not be necessary.

RTS IS NEEDED:

- We must tell log that RTS is needed.
- Some variables are going to be initialized.
- We have to fill all the fields of the RTS frame (duration, addresses …) and we must take care about it is only going to have header (it has no body) and the header length is going to be 16 bytes.
- Once this frame is built we will have in memory two different frames the data one and the RTS one, it is important to keep one copy of the data frame because it is possible that we will need it in the future (for instance, for retransmissions).
- A pointer must point to the beginning of the header.
- A TXVECTOR variable must be created for the RTS frame.
- If the medium in not free: at the time the function is waiting DIFS time it is being tested if the medium is idle in all this DIFS time and if it is not idle backoff function will be called and our function will lose the control until backoff function determinate the STA can transmit. Backoff function will wait until the STA can transmit so, this is a delay until we call PHY_TXSTART_request.
  We tell to LLC that we are unable to send RTS frame at this moment.
- If the medium is free: we must call PHY_TXSTART_request function for transmitting the RTS frame. Here the function loses control.
- Before to call PHY_TXSTART_request we will inform log about the beginning of transmission.

RTS IS NOT NEEDED:

- All the Data frame variables and fields must to be set.
- A TXVECTOR variable must be created for the data frame.
- We send a message to log for saying we get data from LLC.
- Is the medium idle?
- If the medium in not free: at the time the function is waiting SIFS time it is being tested if the medium is idle in all this SIFS time and if it is not idle backoff function will be called and our function will lose the control until backoff function determinate the STA can transmit. Backoff function will wait until the STA can transmit so, this is a delay until we call PHY_TXSTART_request.
  We tell to LLC that we are unable to send RTS frame at this moment.
- If the medium is free: we must actualize some variables and call PHY_TXSTART_request function for transmitting the RTS frame. Here the function loses control.
- Before to call PHY_TXSTART_request we will inform log about the beginning of transmission.

In other words with this function starts the transmission, all the medium access mechanism have to be implemented here. Once the medium is free the transmission begins calling PHY_TXSTART_request and therefore the MAC layer loses the control and gives it to PUM (PHY layer in this case).

4.3.2. backoff

This function is going to be the responsible of decreasing the BackOffTimer of each STA. It will check if the medium is busy at the same time that the BackOffTimer is decreased. When the medium changes to busy the BackOffTimer must be stopped and keep the actual value that it has to continue next time the medium changes to idle (NAV = 0).

Steps in the body of this function:

- First of all this function will check if the BackOffTimer of the STA is 0 or NULL in that case it will be computed a new one. If this BackOffTimer is not 0 it will be decreasing until 0 or until another STA will take the medium (medium busy, NAV != 0), in this case it will stop.
- If the BackOffTimer value is 0, this means that the STA transmitted the last time, therefore :
  - We call random function for getting one new random number, for this we will need the CW value (CW = CWmin in this case, or CW = [(2*CW)+1] if it is a retransmission), we have to take care this value must not be higher than Cwmax
  - This random value is multiplied for the aSlotTime for getting the BackOffTimer. Now the decrease will start.

- When the BackOffTimer is set to 0 BackOff function gives control to the MAC function again and it is time for transmission.

4.3.3. GetRandom

- This function will generate a random number between 0 and CW.
- The value that is going to return is the random number, for this we have to pass this function the CW value.

4.3.4. fduration

This function is going to compute the duration that we must use in each duration field of the frames (RTS, CTS, DATA and ACK). This duration is not going to be the same for every frame, it will depend on the frame type. For computing the duration the medium is going to be busy (this is going to be the information for setting NAV) this fuction is going to need the data length and the frame type. We have to take care about the transmission rate, it is going to be a constant value (54 Mbps)
This function will calculate the duration values in this way:

RTS = 3*SIFS + RTSTime + CTSTime + DataTime + AckTime
CTS = 2*SIFS + CTSTime + DataTime + AckTime
Data = SIFS + DataTime + AckTime
ACK = AckTime
RTSTime = length/rate
DataTime = length/rate
CTSTime = AckTime = length/rate

This function will return the duration value.

4.3.5. PHY_TXSTART_confirm

PHY layer is going to call this function for confirming MAC layer the transmission start. We are going to be waiting this call after calling PHY_TXSTART_request.
When PHY layer calls this function MAC later can transmit the DATA or RTS frame. Now we don't have to take care about the access to the medium mechanism because we have taken care before.

Steps in the body of this function:

- We check the actual STA id (staid) and keep it.
- Log must be called with the TXSTART arriving information.
- A DIFS time must be waited.
- Have we got to send a RTS frame? If we have to send it we will advise log about this. We will initialize all the counters we are going to need before calling PHY_DATA_confirm. Therefore we can call PHY_DATA_confirm.

- If RTS frame doesn't exist we will send DATA frame. For this we must advise log about the DATA transmission. We will initialize all the counters we are going to need before calling PHY_DATA_confirm. Therefore we can call PHY_DATA_confirm.
- The beginning of the transmission in both cases is going to be a PHY_DATA_confirm call. In this call we will pass the frame we want to send and this function will only have to transmit.

4.3.6. PHY_DATA_confirm

This function is going to take charge of sending the frame byte to byte to the PHY layer. The MAC sublayer is going to call this function the first time from PHY_TXSTART_confirm to give this function the frame that is going to be transmited. The PHY layer does next calls to this function each time it transmits one byte. PHY layer is going to call this function as many times as bytes has the sending frame. Once the frame has been sent this function will call PHY_TXEND_request and will lose the control giving it to the PUM.

Steps in the body of this function:

- We check the actual STA id (staid), compare it and keep it.
- We are to have two pointers pointing to the beginning of the header and the body of the frame.
- We will check if we are in the header or we are in the body of the frame comparing the pointer position with the length of the field (these lengths are going to be different in each frame it depends on the frame type).
- We will advise log in each byte we are going to transmit telling it if it is a header frame or a body frame.
- A call to PHY_DATA_request must be done each time we want to transmit one byte.
- When a byte has been sending the pointer must increased one position.
- Once the byte has been sent the PUM will call this function to get the next byte. This mechanism is going to be running until the frame finishes. When this happens log must be advised of the end of the frame and a PHY_TXEND_request call must be done, in this point we lose the control until the PUM calls PHY_TXEND_confirm.

4.3.7. PHY_TXEND_confirm

This function is not going to recive any parameters.

Steps in the body of this function:

- When the PUM calls this function we have to advise log about the end of the transmission.
- Next step is to destroy the frame we have sent and initialice this frame to NULL.
- This functuon will act in several ways depending on the frame type.

- Case RTS:
  - Inform to log about the end of the RTS transmission.
  - Actualizate some variables.
  - It is time for waiting CTSTimeOut and if CTS is received in this time DATA frame must be sent.
- Case DATA:
  - Inform to log about the end of the DATA transmission.
  - Actualizate some variables.
  - It is time for waiting ACKTimeOut and if ACK is received in this time we advise LLC about the success of the transmission.
- Case CTS:
  - Inform to log about the end of the CTS transmission.
- Case ACK:
  - Inform to log about the end of the ACK transmission.

- When this function finish we lose control in the program and PUM get it.

We must take care because we only are going to have these functions for working with all the different kind of frame types. This means that one transmission function must be able to transmit whatever kind of frame, and one reception function must be able to recive whatever kint of frame. The same function that is reciving DATA is going to recive RTS, CTS and ACK, and the same with transmision. We must control this with a state machine changing to transmission and reception each time and calling the responsible functions for this.

4.3.8. RTSTimeout

This function is a time-out between MAC implementation and test bench, we call it using registerTimeOut from PHY_TXEND_confirm (for instance).
At the time that we are waiting CTSTimeOut we are checking if the CTS has come.

CTS has come when the time out expires:

- Its time to set the DATA frame fields.
- A TXVECTOR variable must be created for the data frame.
- We send a message to log for saying we are requesting transmission data start.
- Is the medium idle?
- If the medium in not free: at the time the function is waiting SIFS time it is being tested if the medium is idle in all this SIFS time and if it is not idle BACKOFF function will be called and our function will lose the control until backoff function determinate the STA can transmit. Backoff function will wait until the STA can transmit so, this is a delay until we call PHY_TXSTART_request.
  We tell to LLC that we are unable to send DATA frame at this moment.

- If the medium is free: we must actualize some variables and call PHY_TXSTART_request function for transmitting the RTS frame. Here the function loses control.
- Before to call PHY_TXSTART_request we will inform log about the beginning of transmission.

CTS has not come when the time out expires:

- LLC must be informed using MA_UNITDATA_indicate.

4.3.9. ACKTimeout

This function is a time-out between MAC implementation and test bench, we call it using registerTimeOut from PHY_TXEND_confirm (for instance).
At the time that we are waiting ACKTimeOut we are checking if the ACK has come.

ACK has come when the time out expires:

- LLC must know that everything was right using MA_UNITDATA_indicate.

ACK has not come when the time out expires:

- We must resend the frame only if ACKfailcnt is not greater than MAXACKRETRY. If we have to send this frame we will set all the fields and check if the medium is idle for transmission or not (like before).
- If ACKfailcnt is greater than MAXACKRETRY we advise LLC (using MA_UNITDATA_indicate) and discard the frame.

4.3.10. PHY_RXSTART_indicate

When PHY layer has data for MAC sublayer entity, it will call this function and will tell MAC sublayer the reception is going to start. In this function we have to prepare everything for reciving a frame, and after that we will lose the control.
We have to take care about the four different kinds of frames that this function is going to receive.

Steps in the body of this function:

- This function must be able to differentiate what kind of frame is arriving. For this each frame will have a frame identifier (RTS, CTS, DATA, ACK).
- RTS:
  - We set the frame fields.
  - Advise to log about the RTS start receiving.
- DATA:
  - We set the frame fields.
  - Advise to log about the DATA start receiving.

- CTS:
  - We set the frame fields.
  - Advise to log about the CTS start receiving.
- ACK:
  - We set the frame fields.
  - Advise to log about the ACK start receiving.
- Destroy RXVECTOR that we have received.
- When this function finish the function loses control and gives it to the PUM, but everything is ready for the beginning of the frame reception.

4.3.11. PHY_DATA_indicate

PUM will call PHY_DATA_indicate once for byte of the frame that we are going to receive. This function must to keep these bytes using an array and when PHY layer will not have more bytes to transmit us (the frame is finished) it will call PHY_RXEND_indicate function to indicate the end of the frame. With this call the PHY layer will tell MAC the status of the recited frame.

Steps in the body of this function:

- The STA id must be kept.
- PUM will begin to call this function with the first byte of the frame we are receiving and this function must keep each byte in one array (incrementing the pointer and advising log each time). For this we will have the pointers and the length of the frame we are receiving, in this way we will know if we are going to be in header or body and when the frame reception is finish.
- Once we receive the last byte we will wait to PUM calls PHY_RXEND_indicate.

4.3.12. PHY_RXEND_indicate

When PUM calls this function means that the reception of the frame has finished, therefore we are going to follow several steps:

- The STA id must be kept.
- Addresses must be set.
- What frame type we have received?

Received frame = RTS:

- Addresses comparation must be done for knowing if everything is right.
- If the addresses comparation is right log must be advised about the end of the RTS reception in the STA.
- NAV must be updated.
- All the CTS frame fields must be set.
- Address fields must be copied and duration field must be calculated.
- We take care about CTS is ready for sending and create a txvector for CTS.
- Once CTS frame is ready for sending we must wait SIFS.

- Log must be advised about the CTS transmission.
- MAC calls PHY_DATA_confirm giving to this function the CTS frame we want to transmit and MAC loses control when this function send the first byte.
- If the addresses comparation is not right we advise LLC and give control to the PUM.

Received frame = CTS:

- Addresses comparation must be done for knowing if everything is right.
- If the addresses comparation is right log must be advised about the end of the CTS reception in the STA.
- NAV must be updated.
- A boolean must be set for knowing that CTS has been received (in relation with other functions)
- If the addresses comparation is not right we advise LLC and give control to the PUM.

Received frame = DATA:

- Addresses comparation must be done for knowing if everything is right.
- If the addresses comparation is right log must be advised about the end of the DATA reception in the STA.
- Data frame must be sent to the LLC.
- NAV must be updated.
- All the ACK frame fields must be set.
- Address fields must be copied and duration field must be calculated.
- We take care about ACK is ready for sending and create a txvector for ACK.
- Once ACK frame is ready for sending we must wait SIFS.
- Log must be advised about the ACK transmission.
- MAC calls PHY_DATA_confirm giving to this function the CTS frame we want to transmit and MAC loses control when this function send the first byte.
- If the addresses comparation is not right we advise LLC and give control to the PUM.

When a DATA frame is recibed MAC send it to LLC (PUM in our case but it will take care before that it isnt a retransmited frame. If it is a retransmited frame some counters will increase and the frame could be discarted in the transmission if the counter sets to the maximum value.

Recibed frame = ACK:

- Addresses comparation must be done for knowing if everything is right.
- If the addresses comparation is right log must be advised about the end of the ACK reception in the STA.
- NAV must be updated.
- A boolean must be set for knowing that CTS has been received (in relation with other functions). Some counters must be increased.
- We set NAV to 0 because the transmission is finished.
- If the addresses comparation is not right we advise LLC and give control to the PUM.

4.3.13. PHY_CCA_indicate

PUM (like PHY layer) will call this function each time the medium changes from busy to idle or from idle to busy. Inside it we will actualizate a vble that is going to tell us if the medium is busy or idle. This function is going to be the nucleus of the phisical contention to the medium.

4.3.14. PHY_CCARESET_confirm

When a frame arrives to MA_UNITDATA_request we will implement the access to the medium, if NAV (logical access) tell us the medium is busy, we will keep the frame and will wait to the function PHY_CCARESET_confirm for beginning the transmission.

4.3.15. initialise

This function is not coming from IEEE, it is coming from PUM and we will use it for initialising all the variables and fields that we are going to use for the MAC implementation.

4.3.16. destroy

Empty.

4.3.17. FillSequence

This function fills the sequence number in the header flag of the frame.
When the sequence number is equal to 4096 it starts again from 0 to 4096.

4.3.18. IsMediumBusy

This function returns 0 if the medium is idle or 1 if the medium is busy phisically (state) or logically (NAV).

4.3.19. TestChannelIdleAndBackOff

This function checks if the medium is idle.

If the medium is idle:

- It waits an iTimer (SIFS, DIFS…) and sets some variables.

If the medium is not idle:

- It calls backoff (TestChannelIdleAndBackOffNoWait) and say it to log.

4.3.20. TestChannelIdleAndBackOffNoWait

This functions calls to BackOff when the medium is not idle, it advises log about this.

4.3.21. TimeOutIdle

This function checks the Test state and if it busy it will call the function TestChannelIdleAndBackOffNoWait for BackOff

4.3.22. test

It only shows a message in log file when the test routine is actived.

4.3.23. wait

This function waits times (SIFS, DIFS) using sleep instruction. Wait function has been built for working in windows and unix.

4.3.24. create_MAC

It creates a MAC frame with all the fields that are going to be needed. At first it creates a DATA frame but we will use this frame for control frames which have less fields than DATA frame.

4.4. EXAMPLE OF MAC IMPLEMENTATION

A simple MAC implementation is delivered with the test bench with only the most basic features. This implementation can only communicate using RTS, CTS, ACK and DATA frames while connected to a BSS and using the DFC access protocol. This example has been used for the design of our MAC implementation, trying to improve it using the IEEE standard 802.11 steps.
To see this example go to appendix C.

5. IMPLEMENTATION

5.1 CONFIGURATION FILE

The configuration file holds information about the structure of the simulated network and general information about the test (see background).
Our MAC implementation is going to have 5 stations and each station will be connected with the others by channels. For this reason, each station will have 4 channels for transmission and 4 channels for reception.

Here is the general information:
<general>
nr_of_stations = 5
log_file = "MAClog.log"
log_level = "medium"
</general>

Here is station #1:
<station>
id = 1
name = "Sta1"
test_vector_file = "tvf1.tv"
mac_implementation = "../mac/mac.la"
</station>

Here is station #2:
<station>
id = 2
name = "Sta2"
test_vector_file = "tvf2.tv"
mac_implementation = "../mac/mac.la"
</station>

Here is station #3:
<station>
id = 3
name = "Sta3"
test_vector_file = "tvf3.tv"
mac_implementation = "../mac/mac.la"
</station>

Here is station #4:
<station>
id = 4
name = "Sta4"
test_vector_file = "tvf4.tv"

mac_implementation = "../mac/mac.la"
</station>

Here is station #5:
<station>
id = 5
name = "Sta5"
test_vector_file = "tvf5.tv"
mac_implementation = "../mac/mac.la"
</station>

Here are some channels:
<channels>
station 1 channel [2, "cha"]
station 1 channel [3, "cha"]
station 1 channel [4, "cha"]
station 1 channel [5, "cha"]
station 2 channel [1, "cha"]
station 2 channel [3, "cha"]
station 2 channel [4, "cha"]
station 2 channel [5, "cha"]
station 3 channel [1, "cha"]
station 3 channel [2, "cha"]
station 3 channel [4, "cha"]
station 3 channel [5, "cha"]
station 4 channel [1, "cha"]
station 4 channel [2, "cha"]
station 4 channel [3, "cha"]
station 4 channel [5, "cha"]
station 5 channel [1, "cha"]
station 5 channel [2, "cha"]
station 5 channel [3, "cha"]
station 5 channel [4, "cha"]
</channels>

5.2. MAC IMPLEMENTATION

It must be remembered that this code is the transmission part of the MAC implementation, for seing the reception part go to Implementation of the MAC layer in C languaje writen by Maria Rubio Portales. The functions we can see in the reception part must be added to this code for obtaining the whole MAC implementation.
To see the Transmission part of the MAC implementation go to appendix D.

6. EVALUATION AND RESULTS

In this chapter of the project, the results are going to be evaluated. To analyze the possible errors and what should have been done different is very important to avoid mistakes in future projects. Here are also shown the main problems that have been found during the elaboration of the whole project.

6.1. TECHNICAL EVALUATION

The main goal of this project was to design a MAC sublayer implementation based in the IEEE 802.11 standard specifications which must be able to interact with the test bench where it was going to be simulated. It has not been able to simulate this MAC implementation with the test bench because a problem of time, but all the design is oriented for doing it in the future. This is an obvious reason for not exactly knowing if the design really works in the way it has to do it.
If we do not considerate this problem, the evaluation about the technical view must talks about times for instance, it is known that each PHY layer has its own times, time-out, rates, etc. for this reason all the times in this MAC implementation are computed looking at this OFDM PHY layer characteristics. There are also a lot of maximum or minimum levels that must not being exceeded, this levels are all imposed in the standard.
Is known that this MAC implementation behavior does not answer exactly like the standard specifications ordered, this is because some reasons that are discussed in the following chapter. If the reader studies carefully the IEEE 802.11 standard and this design he will notice the lack of some fields (fragmentation, WEP or CRC for instance), it is due to the fact of the out coming time, for this reason some fields have been shrunk. The reader must also considerate the specifications for this project which are written before at the beginning of the document.
It must be considerate that some ways have been followed for the correct behavior of all the code, this means, that there is some fields that are not exactly like they must to be (the way time-outs are waited for instance).

6.2. PROBLEMS

A few problems has been found during the develop of this project, some problems have been more significant than other:

6.2.1. Test Bench

As mentioned before, the test bench is the environment where this MAC implementation must to be simulated. Due to the lack of time it has not been possible to use this environment, that is the reason because this design is oriented to this test bench but not prove it. To prove the design is essential for knowing how it works, if we considerate this, it has been tried in every moment to suppose as much as possible how the test bench will answer to the code.

It must be known that the MAC implementation code is very large and probably a little complicated for understanding, for this reason it has been tried in every moment to be careful and clear. The C language code has been compilated free of errors.

6.2.2. Specifications

It has been difficult to find a balance between our project specifications, standard specifications and test bench specifications. A lot of problems have appeared due to this, sometimes this problems have been solved shrinking or missing some fields of the standard specifications, always trying to find the better solution for the design.
It has to be considerate that we are working with very extensive standard specifications and with a test bench that we cannot use from the beginning.

6.2.3. Other problems

This project has been done thanks to a scholarship that has allowed me to be abroad my original country. This is an advantage for too many reasons but it also gives you some extra difficulties. Some of these difficulties have been for instance the language, at first it was difficult to understand the IEEE 802.11 standard because of the language and to write this document has not been easy. Another extra difficulty has been the fact of being working with different kinds of operative systems (Macintosh, Unix, Windows).
Otherwise I can say that I feel glad for have had the possibility to improve all these fields, but at first I must confess that it was a wall that I had to knock down.

6.3. TIME PLAN

The time for doing this project from the beginning was four months and a half. A time plan was doing at the beginning but this time plan has suffered some normal changes. It did not take care about some difficulties that has been found during the project realization. At first too much time was used for understanding all the MAC specifications, and it takes more time that I thought to write the C language code. This time plan has been written at the beginning of this document (see 1.3).
After the termination of this project there was, as in every project, some things left that one might want to adjust or complete.

6.4. DEVELOPMENT

This design tries to be flexible for future modifications or changes.
One functionality that may be of interest is that it is another step for studding and improving WLAN systems. It is known that this telecommunication field is changing day by day and this project tries to help to this development. Working together with the test bench it is a simple way to see how MAC environment works.
Linkoping University is working in a final environment where simulate the MAC implementations and where see the results of this test. This tool is really useful for the future of this field, because it allows a simple study, and it is a simple way to analyze the test results.

6.5. ERROR CORRECTION

Due to the length of the C language code and the complexity of the design it has been hard to compile it without errors, otherwise it will be complete with the simulation. It must be considered that this code has been compiled in different compilers and different operative systems, that's the reason it will probably gives errors under some operative systems (it has no errors under Windows). The simulation of this environment will give an overview of the problem for future studies.

6.6. CONCLUSIONS

Some questions can be made regarding this MAC implementation design for instance if it is the correct solution based in the IEEE 802.11 standard.

The current design tries all the time to be as close as possible to the standard specifications but it is quite difficult to find a perfect solution between the standard specifications, PUM specifications and our own specifications for this implementation. This design solves the problem as the problem was presented at start of the project being very close to the standard specifications and interacting with the PUM at the same time.

Finally this design is not more than a C language code that follows some specifications and has to interact with other C language code files.

In this way, there is a compromise between the specifications and the other codes that has been solved as better as possible trying always to find a flexible solution. One of the final conclusions from the author is that the code must be as flexible as possible due to the complex for finding a good relationship between standard specifications and PUM specifications. If the code is flexible it gives an improving possibility in the future if it is necessary. Fortunately one of the best characteristic of C language is the flexibility that it provides.

Another conclusion is that MAC layer is a unique entity and it must be regarded like that. It is complicated to try to divide the problem in two parts, transmission and reception because they are both mixed in all the implementation. The problem begins with the questions, where does the transmission/reception start? Or, where does the transmission/reception finish? For this MAC layer must be considerate like a whole entity and it has no sense to read the transmission part without reading the reception part.

7. REFERENCES

These are the references of sources where information has been found.

- ANSI/IEEE Standard 802.11, Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,
  1999 Edition,
  Piscatawa, USA : IEEE Standards Board. ISBN 0–7381–1658–0.

- Principles of Wireless Networks,
  Prashant Krishnamurthy, Kaveh Pahlavan,
  2001-21-11,
  ISBN: 0130930032

- Comprehensive C,
  David A. Spuler,
  Prentice Hall, 1992 Edition,
  ISBN 0-13-156514-1.

- User manual, MAC test bench for IEEE 802.11a, pum10_. Ver 1.0,
  Felix Lundmark,
  2003-05-12,
  Linkoping Institute of Technology, Dept. of Computer and Information Science.

- Computer Networks,
  A.S. Tanenbaum,
  Prentice Hall, 2003

- Redes Inalámbricas de Área Local,
  Alexis Dowhuszko – Maximiliano Eschoyez – Matias Freytes,
  Facultad Ciencias Exactas Fisicas y Naturales,
  Universidad Nacional de Cordoba,
  lcd.efn.unc.edu.ar/frames/archivos/fernets.pdf

- Introduccion a las Redes Inalambricas 802.11,
  Javier Canas R.,
  2003-03-13

- Physical layer interface for IEEE 802.11 MAC,
  Per Neren,
  2002-10-01,
  Linkoping University,
  Web: www.ep.liu.se/exjobb/isy/2002/3241/

APPENDIX A: Abbreviations and acronyms

ACK	acknowledgment
AP	access point
ATIM	announcement traffic indication message
BSA	basic service area
BSS	basic service set
CCA	clear channel assessment
CF	contention free
CFP	contention-free period
CP	contention period
CRC	cyclic redundancy code
CS	carrier sense
CSMA/CA	collision avoidance medium sharing mechanism
CW	contention window
DA	destination address
DCF	distributed coordination function
DIFS	distributed (coordination function) interframe space
DLL	data link layer
DSS	distribution system service
EIFS	extended interframe space
FC	frame control
FCS	frame check sequence
IBSS	independent basic service set
IFS	interframe space
IV	initialization vector
LAN	local area network
LLC	logical link control
LME	layer management entity
LRC	long retry count
MAC	medium access control
MIB	management information base
MLME MAC	sublayer management entity
MPDU MAC	protocol data unit
N/A	not applicable
NAV	network allocation vector
PC	point coordinator
PCF	point coordination function
PDU	protocol data unit
PHY	physical (layer)
PIFS	point (coordination function) interframe space
PLCP	physical layer convergence protocol
PLME	physical layer management entity
PS	power save (mode)
RA	receiver address

RTS	request to send
RX	receive or receiver
SA	source address
SIFS	short interframe space
SLRC	station long retry count
SRC	short retry count
SSRC	station short retry count
STA	station
TA	transmitter address
TU	time unit
TX	transmit or transmitter
WAN	wide area network
WEP	wired equivalent privacy
WLAN	wireless local area network

APPENDIX B: Definitions, variables, times and constants

Here are some definitions, variables, times and constants needed for understanding the document or used in the C language MAC implementation.

TX	Transmission station
RX	Reception station
MedState	1 means medium is busy, 0 means medium is free
NAV	(Network allocation vector) It updates with RTS/CTS and headers information, it shows the time the medium is going to be busy, the medium will be free when NAV value is 0 (NAV=0 idle, NAV=no 0 busy)
BackOffTime	Random()*aSlotTime.
Random()	Random interval between [0,CW]
aSlotTime	BackOff Timer will be built by several slots, this slots are a constant value of time. .
SIFS	(Short IFS) We will use it with ACK, CTS and fragments. It is 16 μs
DIFS	(DCF IFS) We will use it with Data Frames y Backoff. It is 34 μs
EIFS	(Extended IFS) We will use it when the last transmitted frame has been wrong, it will not wait carrier virtual sense.
CWmin	Minimum value of the contention window. This value is equal to 15.
CWmax	Maximum value of the contention window. This value is equal to 1023.
CW	Value between the minimum and the maximum it will increase each time that SSRC o SLRC increase. It increases in this way {2*(CW)+1} the first value of CW will be CWmin and it will never go upper than CWmax. CW will reset to CWmin each time the frame will transmit ok or when SSRC or SLRC take the values SRL or LRL
SSRC (STA)	It increases when we do not receive ACK in a frame lower than RTSThreshold (without RTS). It resets to 0 when it receives CTS or when it receives an ACK
SLRC (STA)	It increases when we do not receive ACK in a frame higher than RTSThreshold (with RTS). It only resets to 0 when it receives ACK
SRC (MSDU)	It increases when we do not receive ACK in a frame lower than RTSThreshold (without RTS). It resets to 0 when it receives CTS or when it receives an ACK
LRC (MSDU)	It increases when we do not receive ACK in a frame higher than RTSThreshold (with RTS). It only resets to 0 when it receives ACK
SRL	(Short retry limit) It controls the times that a frame lower than RTS Threshold will be transmitted without receiving ACK before thinking the transmission has failed. This limit is 7.(For SRC and SSRC)
LRL	(Long retry limit) It controls the times that a frame equal or upper than RTS Threshold will be transmitted without receiving ACK before thinking the transmission has failed. This limit is 4.(For LRC and SLRC)
MaxTxMSDULifeTime	It controls and limits the time from the beginning of the transmission to the end. Its value is 524 ms (524000 μs)

MaxRxLifeTime	It limits the time that the STA is going to be waiting to the frame. Its value is 512 ms (512000 μs)
MPDUMaxLength	This is a constant value equal to 2346 octets. It is going to be the maximum length of the MPDUs.
FailCnt	It counts the number of failed frames due to be upper than SRL or LRL
RetryCnt	It counts the number of sent frames needed or a retransmission
MulRetryCnt	It counts the number of frames that have needed a retransmission
FrameDupliCnt	It counts the times that the station receives duplicate frames
RTSSuccessCnt	It increases each time we receive CTS when RTS has been sent
RTSFailCnt	It increases each time we do not receive CTS when RTS has been sent
AckFailCnt	It increases each time we do not receive ACK
FCSTxFrameCnt	It increases each time we receive a correct MSDU
AckResponseTimeOut	This is the time we are going to wait ACK after transmit a frame, if this time expires the transmission has failed.
OpRateSet	It shows the list of transmission rates for each BSS
BeaconPeriod	It controls the time between target beacon transmission times
PrivacyInvoked	(Boolean) It will say if we have used WEP
ExcludeUnencrypted	(Boolean) It shows when the data is sent without WEP
ExUnencryptedCnt	This counter increases each time WEP is not used.
MACAddress	MAC address. It will be used for sending frames to the upper layers (48 bits)
RTSThreshold	When the length of the frame is upper than this value we will always use RTS/CTS
ACKb	Ack boolean (true or false). This value depends on if the ACK has been received successfully or not
RyCb	Rts/Cts Boolean. This value depends on if the CTS has been received or not
CTSTimeOut	This is the maximum time that we are going to be waiting CTS before thinking it has been lost.

MSDU generic DATA frame flags (the values are in bits)

*P*roVer	This value is always going to be 00 (protocol version)
Type	This value is always going to be 10 for data
SbType	Always 0000 for being a data frame
ToDS	It will be 0 for being a data frame and not going to DS
FromDS	It will be 0 because it not comes from DS
MoreFrag	It is 1 when there are more fragments to transmit or 0 when there are not more fragments or when the frame does not need to be fragmentized
Retry	It is 1 the frame is being retransmitted or 0 when it is the first transmission
PwrMgt	It is 1 when it is in save power, or 0 in active mode (the STA will be in active mode for receiving a frame)

MoreData	It shows when it exists more data waiting in queue for being transmitted. Only when the STA is in power save
WEP	It is 1 when WEP is used or 0 when it is not used
ID	It shows the frame transmission address, they are 16 bits, from 15 to 0, the others will take one value from 0 to 32767 it depends on the duration
DA	Destiny address
SA	Source address
RA	Reception address
TA	Transmission address
BSSID	It shows the IBSS where we are
Addr1	This is the first frame address flag and it will be equal to RA=DA when it is a data frame
Addr2	This is the second frame address flag and it will be equal to SA when it is a data frame
Addr3	This is the third frame address flag and it will be equal to BSSID when it is a data frame
Addr4	This is the third frame address flag and it will be equal to N/A when it is a data frame
SeqNum	It shows the frame number that we are transmitting

RTS/CTS and ACK (this values are in bits)

DurRTS	It shows the time that the medium is going to be busy from RTS is sent to ACK is received
DurCTS	It shows the time that the medium is going to be busy from CTS is sent to ACK is received
RArts	It shows the receiver RTS address
TArts	It shows the sender RTS address
RActs	Is going to be equal to RArts
RAack	This value is going to be a copy of Addr2 (SA)

Other especial variables used in the MAC implementation

HEADER_SIZE	It is the data header sizeequal to 30 bytes
HEADER_SIZE_RTS	It is the RTS header sizeequal to 16 bytes
HEADER_SIZE_CTS	It is the CTS header sizeequal to 10 bytes
HEADER_SIZE_ACK	It is the ACK header sizeequal to 10 bytes
header_pos	It will point to a header possition
header_size	Size of the header
data_pos	It will point to a data possition
data_size	Size of the data
datalen	Length of the data
MSG_MAX_SIZE	Maximum size of the message
MAXACKRETRY	Maximum number of times one ACK can be sent

cur_par	Defines the parts of the frame, header and body
CTSrsp	Have we received CTS at time?
ACKrsp	Have we received ACK at time?
RetryCnt	It counts the number of times a frame has been retransmitted
CTScount	It counts the times that a STA receives ACK frame
ACKcount	It counts the times that a STA receives ACK frame
BackOffTimer	This timer is used in backoff procedure. It will decrease until reach 0, in this moment the STA is allowed to transmit
Frametype	Type of frame. It can be RTS, CTS, DATA or ACK
FrameControl	In this field are defined all the flags in the frame control (in bits)
Duration	Used for knowing the time that one transmission is going to have the medium busy
Maccopydata	Used for copping the data frame
bIdleTest	Boolean used for knowing if the test is idle or not
bReceivedCTS	Boolean used for knowing if CTS has been received
bReceivedACK	Boolean used for knowing if ACK has been received
RTSneed	Used for knowing if RTS is needed

There are some parametres used in the way the PUM says for having a good conection between Test Bench and MAC implementation, these parametres are shown in the Test Bench manual (PUM)

APPENDIX C: Example of MAC implementation

This is an example of MAC implementation in C language. (See 4.4)

// file *Example of MAC implementation code*.

```
#include <stdlib.h>
#include <stdio.h>
#include "mactypes.h"

#define BOOL short
#define TRUE 1
#define FALSE 0
#define HEADER_SIZE 30
#define MSG_MAX_SZ 160

enum cur_part { header, body};

struct MAC
{
 // Save the setup data here
 MAC_SETUP* setup;
 octet_t* data;
 unsigned int data_pos;
 unsigned int data_size;
 // Header for data frames
 octet_t header[30];
 unsigned int header_pos;
 enum cur_part part;
 CCASTATE state;
 BOOL waiting;
 char* message;
};

/*
 struct MACFrameHeader {

 octet_t frameControl[2];
 octet_t durationOrID[2];
 octet_t addr1[6];
 octet_t addr2[6];
 octet_t addr3[6];
 octet_t sequenceControl[2];
 octet_t addr4[6];
 };
```

```
 struct MACFrame
{
 struct MACFrameHeader* mh;
 octet_t data;
 };

/* ================================================================
 * Function declarations (prototypes)
================================================================*/

MAC* create_MAC(void);

void initialise_MAC(MAC* mac, MAC_SETUP* macs);

void destroy_MAC(MAC* mac);

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data);

void PHY_DATA_confirm(MAC* mac);

void PHY_TXSTART_confirm(MAC* mac);

void PHY_TXEND_confirm(MAC* mac);

void PHY_CCARESET_confirm(MAC* mac);

void PHY_DATA_indicate(MAC* mac, octet_t data);

void PHY_CCA_indicate(MAC* mac, CCASTATE state);

void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv);

void PHY_RXEND_indicate(MAC* mac, RXERROR rxe);

/* ================================================================ */

void setAddr1(macaddr_t* a, octet_t* header)
{
 header[4] = a->addr[0];
 header[5] = a->addr[1];
 header[6] = a->addr[2];
 header[7] = a->addr[3];
 header[8] = a->addr[4];
 header[9] = a->addr[5];
}
```

```
void setAddr2(macaddr_t* a, octet_t* header)
{
  header[10] = a->addr[0];
  header[11] = a->addr[1];
  header[12] = a->addr[2];
  header[13] = a->addr[3];
  header[14] = a->addr[4];
  header[15] = a->addr[5];
}

void getAddr1(macaddr_t* a, octet_t* header)
{
  a->addr[0] = header[4];
  a->addr[1] = header[5];
  a->addr[2] = header[6];
  a->addr[3] = header[7];
  a->addr[4] = header[8];
  a->addr[5] = header[9];
}

void getAddr2(macaddr_t* a, octet_t* header)
{
  a->addr[0] = header[10];
  a->addr[1] = header[11];
  a->addr[2] = header[12];
  a->addr[3] = header[13];
  a->addr[4] = header[14];
  a->addr[5] = header[15];
}

void test(MAC*, int);

MAC* create_MAC(void)
{
  MAC* mac = (MAC*)malloc(sizeof(MAC));
  if (mac == NULL) {
    fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
    abort();
  }
  mac->setup = NULL;
  mac->data = NULL;
  mac->data_pos = 0;
  mac->data_size = 0;
  mac->state = cca_idle;
  mac->waiting = FALSE;
  mac->message = malloc(MSG_MAX_SZ);
```

```
 return mac;
}

void initialise_MAC(MAC* mac, MAC_SETUP* macs)
{
 if (mac->setup != NULL)
 {
   destroy_MAC_SETUP(mac->setup);
 }
 mac->setup = macs;
}

void destroy_MAC(MAC* mac)
{
 if (mac != NULL)
 {
   destroy_MAC_SETUP(mac->setup);
   destroyData(mac->data);
   free(mac->message);
   free(mac);
  }
}

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data)
{

 // Receive data from LLC
 // We'll start by creating a valid frame header for the data

 // Test code
 int id = mac->setup->staid;
 mac->data_pos = 0;
 mac->data_size = data->datalen;
 mac->part = header;
 mac->header_pos = 0;
 if (!macaddrCompare(&mac->setup->macaddr ,&data->saddr) ) {
   fprintf(stderr, "MAC %d: Incorrect source address\n", id);
   abort();
 }
 if (data->datalen > 2304)
 {
   fprintf(stderr, "MAC %d: Got data unit of illegal length %u. Aborting.\n",
           id, data->datalen);
   abort();
 }
```

```
  // Set header fields
  setAddr1(&data->daddr, &mac->header[0]);
  setAddr2(&data->saddr, &mac->header[0]);

  // Create some test data
  mac->data = data->data;
  snprintf(mac->message, (size_t)MSG_MAX_SZ,
           "Got data unit of length %d from LLC.", mac->data_size);
  mac->setup->log(id, mac->message, lp_low);
  TXVECTOR* txv = create_TXVECTOR(mac->data_size + HEADER_SIZE);
  snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission start.");
  mac->setup->log(id, mac->message, lp_low);
  mac->setup->PHY_TXSTART_request(id,txv);
}

void PHY_DATA_confirm(MAC* mac)
{
  const int id = mac->setup->staid;
  mtime_t time = mac->setup->getTime(id);
  if (mac->part == header) {
    // Send header
    if (mac->header_pos < HEADER_SIZE) {
      snprintf(mac->message,(size_t)MSG_MAX_SZ,
               "Header octet %d/%d sent: 0x%.2hhX",
               mac->header_pos, (HEADER_SIZE - 1),
               mac->header[mac->header_pos]);
      mac->setup->log(id, mac->message, lp_debug);
      mac->setup->PHY_DATA_request(id, mac->header[mac->header_pos]);
      ++(mac->header_pos);
    }
    else
    {
      mac->part = body;
    }
  }
  if (mac->part == body)
  {
    // Send body
    if (mac->data_pos < mac->data_size) {
      snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data octet %d/%d sent:
0x%.2hhX",
               mac->data_pos, (mac->data_size - 1),
               mac->data[mac->data_pos]);
      mac->setup->log(id, mac->message, lp_debug);
      mac->setup->PHY_DATA_request(id, mac->data[mac->data_pos]);
      ++(mac->data_pos);
```

```
    }
    else
    {
      snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission end.");
      mac->setup->log(id, mac->message, lp_low);
      mac->setup->PHY_TXEND_request(id);
    }
  }
}

void PHY_TXSTART_confirm(MAC* mac)
{
  // Test code - Start sending
  const int id = mac->setup->staid;
  snprintf(mac->message,(size_t)MSG_MAX_SZ,"Got confirmation of TXSTART");
  mac->setup->log(id, mac->message, lp_low);
  PHY_DATA_confirm(mac);
}

void PHY_TXEND_confirm(MAC* mac)
{
  destroyData(mac->data);
  mac->data = NULL;
}

void PHY_CCARESET_confirm(MAC* mac)
{
}

void PHY_DATA_indicate(MAC* mac, octet_t data)
{
  // Receive data
  int id = mac->setup->staid;
  mtime_t time = mac->setup->getTime(id);
  if (mac->part == header)
  {
    if (mac->header_pos < HEADER_SIZE)
    {
      mac->header[mac->header_pos] = data;
      snprintf(mac->message,(size_t)MSG_MAX_SZ,"Header octet %d/%d recieved:
0x%.2hhX",
               mac->header_pos, (HEADER_SIZE - 1),
               mac->header[mac->header_pos]);
      mac->setup->log(id, mac->message, lp_debug);
      ++mac->header_pos;
    }
```

```
   else
   {
    mac->part = body;
   }
  }
  if (mac->part == body)
  {
   mac->data[mac->data_pos] = data;
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data octet %d/%d recieved:
0x%.2hhX",
            mac->data_pos, (mac->data_size - 1),
            mac->data[mac->data_pos]);
   mac->setup->log(id, mac->message, lp_debug);
   ++mac->data_pos;
  }
}

void PHY_CCA_indicate(MAC* mac, CCASTATE state)
{
 // Receive indication of PHY state change
 mac->state = state;
 if (state == cca_idle)
 {
  // Start timer
 }
else
{ // cca_busy}
}

void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv)
{
 // Start receiving transmission
 mac->data_pos = 0;
 mac->data_size = rxv->length - HEADER_SIZE;
 mac->data = createData(mac->data_size);
 mac->header_pos = 0;
 mac->part = header;
 destroy_RXVECTOR(rxv);
}

void PHY_RXEND_indicate(MAC* mac, RXERROR rxe)
{
 // End receiving transmission
 int id = mac->setup->staid;
 mtime_t time = mac->setup->getTime(id);
 mac->setup->log(id,"Finished recieving data unit.",lp_low);
```

```
  macaddr_t saddr;
  macaddr_t daddr;
  // Get header fields
  getAddr1(&daddr, &mac->header[0]);
  getAddr2(&saddr, &mac->header[0]);

  if (macaddrCompare(&daddr, &mac->setup->macaddr))
  {
    MA_UNITDATA* mad = create_MA_UNITDATA(saddr, daddr, mac->data,
                                          mac->data_size, contention,
                                          reorderable_multicast);
    mac->data = NULL;
    mac->setup->log(id,"Data unit recieved.",lp_medium);
    mac->setup->MA_UNITDATA_indicate(id, mad);
   }
}

void test(MAC* mac, int arg)
{
  int id = mac->setup->staid;
  snprintf(mac->message,(size_t)MSG_MAX_SZ,
           "Test routine activated, argument = %d", arg);
}
```

APPENDIX D: Transmission part

In this appendix is the Transmission part of the MAC implementation design. It can also be interesting for the reader to read the Reception part of this MAC implementation, for understandig the whole design.

```
// This file contains the MAC implementation code.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mactypes.h"
#include <windows.h>

#define snprintf _snprintf
#define BOOL short
#define TRUE 1
#define FALSE 0
#define HEADER_SIZE 30
#define HEADER_SIZE_RTS 16
#define HEADER_SIZE_CTS 10
#define HEADER_SIZE_ACK 10
#define MSG_MAX_SZ 160
#define RTSThreshold 2000          /* We can change this */
#define aSlotTime 9000             /* These values are in nanoseconds */
#define SIFS 16000
#define DIFS 34000
#define CWMIN 15                   /* CW values */
#define CWMAX 1023
#define CTSTimeOut 19852           /* 26*8/54*1000+SIFS */
#define ACKTimeOut 363259
#define MAXACKRETRY 16

enum cur_part {header, body};
unsigned int CTSrsp;
unsigned int ACKrsp;
unsigned int ACKb = 0;
unsigned int RetryCnt = 0;
unsigned int CTScount = 0;
unsigned int ACKcount = 0;
unsigned int RTSSuccessCnt = 0;
unsigned int RTSFailCnt = 0;


/*This is going to be the struct for Data frames */

enum Tframetype {RTS = 1, CTS = 2, DATA = 3, ACK = 4};
struct MAC
{
```

```
// Vbles para la contienda
unsigned int CW;
unsigned int BackOffTimer;
// Este va a ser el id de trama
Tframetype frametype;
// Save the setup data here
MAC_SETUP* setup;
octet_t* data;
unsigned int data_pos;
unsigned int data_size;
// Header for data frames
// We use uHeader.header to get a byte array
// We use uHeader.sHeader.XXXX to get an struct with the same format that the frame,
// using a bit field for the frame control field.
union TuHeader
{
        octet_t header[30];
        struct
        {
         struct
         {
          unsigned short
          ProtocolVersion:2,
           Type:2,
          SubType:4,
          ToDS:1,
           FromDS:1,
          MoreFrag:1,
          Retry:1,
          PWRMgnt:1,
          MoreData:1,
          WEP:1,
          Order:1;
         } FrameControl;
   unsigned short duration;
   macaddr_t Address1;
   macaddr_t Address2;
        } sHeader;
} uHeader;
unsigned int header_pos;
unsigned int header_size;
enum cur_part part;
unsigned int duration;
CCASTATE state;
BOOL waiting;
char* message;
MA_UNITDATA *maccopydata; // This var is used to save a data frame when we send an RTS
                          // and we're waiting to receive the CTS.
unsigned int SeqContNum;
unsigned int NAV;
BOOL bIdleTest;
```

```
  BOOL bReceivedCTS;
  BOOL bReceivedACK;
  unsigned int ACKfailCnt;
};

/* Got dat */
  struct MACFrameHeader
  {
  octet_t frameControl[2];
  octet_t durationOrID[2];
  octet_t addr1[6];
  octet_t addr2[6];
  octet_t addr3[6];
  octet_t sequenceControl[2];
  octet_t addr4[6];
  };

  struct MACFrame
  {
  struct MACFrameHeader* mh;
  octet_t *data;
  };


/* ============================================================
 * Function declarations (prototypes)
 ============================================================*/

MAC* create_MAC(void);

void initialise_MAC(MAC* mac, MAC_SETUP* macs);

void destroy_MAC(MAC* mac);

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data);

void PHY_DATA_confirm(MAC* mac);

void PHY_TXSTART_confirm(MAC* mac);

void PHY_TXEND_confirm(MAC* mac);

void PHY_CCARESET_confirm(MAC* mac);

void PHY_DATA_indicate(MAC* mac, octet_t data);

void PHY_CCA_indicate(MAC* mac, CCASTATE state);

void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv);

void PHY_RXEND_indicate(MAC* mac, RXERROR rxe);
```

```
int fduration (Tframetype frametype, int lenght);

BOOL backoff (MAC *mac,unsigned int a, unsigned int b, int c);

int GetRandomNumber(int iMaximum);

void wait(int iTime);

void FillSequence(MAC *mac);

BOOL IsMediumBusy(MAC *mac);

BOOL TestChannelIdleAndBackOff(MAC *mac,int iTimer);

BOOL TestChannelIdleAndBackOffNoWait(MAC *mac);

void RTSTimeout(MAC *mac,int iValue);

void ACKTimeout(MAC *mac,int iValue);

void TimeoutIdle(MAC *mac,int iValue);

/* ================================================================*/

/* These functions are going to set mac addresses in the header of the frames */

void setAddr1(macaddr_t* a, octet_t* header)
{
  header[4] = a->addr[0];
  header[5] = a->addr[1];
  header[6] = a->addr[2];
  header[7] = a->addr[3];
  header[8] = a->addr[4];
  header[9] = a->addr[5];
}

void setAddr2(macaddr_t* a, octet_t* header)
{
  header[10] = a->addr[0];
  header[11] = a->addr[1];
  header[12] = a->addr[2];
  header[13] = a->addr[3];
  header[14] = a->addr[4];
  header[15] = a->addr[5];
}
```

```
/*These functions are going to get mac addresses from the header of the frames*/

void getAddr1(macaddr_t* a, octet_t* header)
{
 a->addr[0] = header[4];
 a->addr[1] = header[5];
 a->addr[2] = header[6];
 a->addr[3] = header[7];
 a->addr[4] = header[8];
 a->addr[5] = header[9];
}

void getAddr2(macaddr_t* a, octet_t* header)
{
 a->addr[0] = header[10];
 a->addr[1] = header[11];
 a->addr[2] = header[12];
 a->addr[3] = header[13];
 a->addr[4] = header[14];
 a->addr[5] = header[15];
}


void test(MAC*, int);

MAC* create_MAC(void)
{
 MAC* mac = (MAC*)malloc(sizeof(MAC));
 if (mac == NULL)
 {
  fprintf(stderr, "Out of memory in %s", "create_MAC");
  abort();
 }
 mac->setup = NULL;
 mac->data = NULL;
 mac->duration = NULL;
 mac->data_pos = 0;
 mac->data_size = 0;
 mac->state = cca_idle;
 mac->waiting = FALSE;
 mac->message = (char*)malloc(MSG_MAX_SZ);
 mac->maccopydata=NULL;
 mac->CW=0;
 mac->BackOffTimer=0;
 mac->frametype=DATA;
 memset(&mac->uHeader.sHeader,0,sizeof(mac->uHeader.sHeader));
 mac->header_pos=0;
 mac->header_size=0;
 mac->part=header;
 mac->SeqContNum=0;
 mac->NAV=0;
```

```
 mac->uHeader.sHeader.FrameControl.ProtocolVersion=0;
 mac->uHeader.sHeader.FrameControl.ToDS=0;
 mac->uHeader.sHeader.FrameControl.FromDS=0;
 mac->uHeader.sHeader.FrameControl.MoreFrag=0;
 mac->uHeader.sHeader.FrameControl.Retry=0;
 mac->uHeader.sHeader.FrameControl.PWRMgnt=0;
 mac->uHeader.sHeader.FrameControl.MoreData=0;
 mac->uHeader.sHeader.FrameControl.WEP=0;
 mac->uHeader.sHeader.FrameControl.Order=0;
 mac->bIdleTest=0;
 mac->bReceivedCTS=0;
 mac->bReceivedACK=0;
 mac->ACKfailCnt=0;
 return mac;
}

void initialise_MAC(MAC* mac, MAC_SETUP* macs)
{
 if (mac->setup != NULL)
 {
  destroy_MAC_SETUP(mac->setup);
 }
 mac->setup = macs;
}

void destroy_MAC(MAC* mac)
{
 if (mac != NULL)
 {
  destroy_MAC_SETUP(mac->setup);
  destroyData(mac->data);
  free(mac->message);
  free(mac);
 }
}

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data)
{
 // Receive data from LLC
 // Test code
 int id = mac->setup->staid;
 mac->CW = CWMIN;
 mac->BackOffTimer = 0;
 if (!macaddrCompare(&mac->setup->macaddr, &data->saddr) )
 {
  fprintf(stderr, "MAC %d: Incorrect source address\n", id);
  abort();
 }
 if ((data->datalen > 2304)||(data->datalen <= 0))
 {
  fprintf(stderr, "MAC %d: Got data unit of illegal length %u. Aborting.\n",
```

```
            id, data->datalen);
   abort();
  }

  // Set header fields
  setAddr1(&data->daddr, &mac->uHeader.header[0]);
  setAddr2(&data->saddr, &mac->uHeader.header[0]);
  FillSequence(mac);

  // Are we going to need RTS?
  if (data->datalen >= RTSThreshold)
  {
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"RTS is needed.");
   mac->setup->log(id, mac->message, lp_low);
   int RTSneed = 1;
   mac->maccopydata=data;
 // RTS must be created
   mac->frametype = RTS;
   mac->data_pos = 0;
   mac->data_size = 0;     /* In RTS data length is 0 */
   mac->part = header;
   mac->header_pos = 0;
   mac->header_size = HEADER_SIZE_RTS;
   mac->duration = fduration (mac->frametype, mac->data_size);
   mac->uHeader.sHeader.duration=mac->duration;
         mac->uHeader.sHeader.FrameControl.Type=1;
         mac->uHeader.sHeader.FrameControl.SubType=11;
   TXVECTOR* txv = create_TXVECTOR(HEADER_SIZE_RTS);
   mac->BackOffTimer = 0;
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"Medium is idle, requesting transmission RTS
start.");
   mac->setup->log(id, mac->message, lp_low);
   if (!TestChannelIdleAndBackOff(mac,DIFS))
   {
          // We were unable to send the RTS. Tell it to the LLC.
           status.tstat=undeliv_time;  // XXXXXX
           status.saddr=mac->uHeader.sHeader.Address1;
           status.daddr=mac->uHeader.sHeader.Address2;
           status.prov_prio=data->prio;
           status.prov_sclass=data->sclass;
           mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
   }
    else
   {
         snprintf(mac->message, (size_t)MSG_MAX_SZ,
           "RTS TX_START request.", mac->data_size);
         mac->setup->log(id, mac->message, lp_low);
         mac->setup->PHY_TXSTART_request(id,txv);
    }
  }
  else // This must also be in the receiver when we receive a CTS.
```

```
  {
   mac->frametype = DATA;
   mac->data_pos = 0;
   mac->data_size = data->datalen;
   mac->part = header;
   mac->header_pos = 0;
   mac->header_size = HEADER_SIZE;
   mac->duration = fduration (mac->frametype, mac->data_size);
   mac->uHeader.sHeader.duration=mac->duration;
   mac->uHeader.sHeader.FrameControl.Type=2;
   mac->uHeader.sHeader.FrameControl.SubType=0;
   mac->data = data->data;
   snprintf(mac->message, (size_t)MSG_MAX_SZ,
            "Got data unit of length %d from LLC.", mac->data_size);
   mac->setup->log(id, mac->message, lp_low);
   TXVECTOR* txv = create_TXVECTOR(mac->data_size + HEADER_SIZE);
   mac->BackOffTimer = 0;  /* Lo ponemos a 0 aunque no deberia de ser necesario */
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission Data start.");
   mac->setup->log(id, mac->message, lp_low);
   if (!TestChannelIdleAndBackOff(mac,SIFS))
   {
   // We were unable to send the DATA frame. Tell it to the LLC.
            status.tstat=undeliv_time;  // XXXXXX
            status.saddr=mac->uHeader.sHeader.Address1;
            status.daddr=mac->uHeader.sHeader.Address2;
            status.prov_prio=data->prio;
            status.prov_sclass=data->sclass;
            mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
    }
    else
    {
          mac->ACKfailCnt=0;
          mac->bReceivedACK=0;
          snprintf(mac->message, (size_t)MSG_MAX_SZ,
            "DATA TX_START request.", mac->data_size);
          mac->setup->log(id, mac->message, lp_low);
          mac->setup->PHY_TXSTART_request(id,txv);
    }
   }
}

void PHY_TXSTART_confirm(MAC* mac)
{
  // Test code - Start sending
  const int id = mac->setup->staid;
  snprintf(mac->message,(size_t)MSG_MAX_SZ,"Got confirmation of TXSTART");
  mac->setup->log(id, mac->message, lp_low);
 // RTS CASE
  if (mac->frametype == RTS)
  {
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"RTS Transmission start");
```

```
  mac->setup->log(id, mac->message, lp_low);
  PHY_DATA_confirm(mac);
 }
 // DATA CASE
 else if (mac->frametype==DATA)
 {
  snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data Transmission start");
  mac->setup->log(id, mac->message, lp_low);
  PHY_DATA_confirm(mac);
 }
}

void PHY_DATA_confirm(MAC* mac)
{
 const int id = mac->setup->staid;
 mtime_t time = mac->setup->getTime(id);
 if (mac->part == header)
 {
  // Send header
  if (mac->header_pos < mac->header_size)
  {
   snprintf(mac->message,(size_t)MSG_MAX_SZ,
            "Header octet %d/%d sent: 0x%.2hhX",
            mac->header_pos, (HEADER_SIZE - 1),
            mac->uHeader.header[mac->header_pos]);
   mac->setup->log(id, mac->message, lp_debug);
   mac->setup->PHY_DATA_request(id, mac->uHeader.header[mac->header_pos]);
   ++(mac->header_pos);
  }
  else
  {
   mac->part = body;
  }
 }
 if (mac->part == body)
 {
  // Send body
  if (mac->data_pos < mac->data_size)
  {
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data octet %d/%d sent: 0x%.2hhX",
            mac->data_pos, (mac->data_size - 1),
            mac->data[mac->data_pos]);
   mac->setup->log(id, mac->message, lp_debug);
   mac->setup->PHY_DATA_request(id, mac->data[mac->data_pos]);
   ++(mac->data_pos);
  }
  else
  {
   snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission end.");
   mac->setup->log(id, mac->message, lp_low);
   mac->setup->PHY_TXEND_request(id);
```

```
  }
 }
}

void PHY_TXEND_confirm(MAC* mac)
{
 const int id = mac->setup->staid;
 snprintf(mac->message,(size_t)MSG_MAX_SZ,"Transmission end confirmation.");
 mac->setup->log(id, mac->message, lp_low);
 destroyData(mac->data);
 mac->data = NULL;
 switch(mac->frametype)
 {
 case RTS:
        // We must wait the reception of CTS to continue sending the data frame.
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"RTS sending finished.");
       // Register a timeout and wait the CTS. Tell the receiver that we've launched
       // the timeout, and setup a variable which we can use to make the receiver to tell us
       // that we've received the CTS which we will test in the timeout function.
        mac->bReceivedCTS=0;
        mac->setup->registerTimeOut(id,CTSTimeOut,RTSTimeout,0);
        break;
 case DATA:
        // We must wait for the reception of ACK to update LLC sending state of this frame.
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data sending finished.");
        // Register a timeout and wait the ACK. Tell the receiver that we've launched
        // the timeout, and setup a variable which we can use to make the receiver to tell us
        // that we've received the ACK which we will test in the timeout function.
        mac->bReceivedACK=0;
        mac->setup->registerTimeOut(id,ACKTimeOut,ACKTimeout,0);
        break;
 case CTS:
        // We have sent CTS frame. No more ops required (NAV update ? ).
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data sending finished.");
        break;
 case ACK:
        // We have sent an ACK. No more ops required.
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"ACK sending finished.");
        break;
 }
 mac->setup->log(id, mac->message, lp_low);
}

void RTSTimeout(MAC *mac,int iValue)
{
 const int id = mac->setup->staid;
 MA_UNITDATA_STATUS status;
 MA_UNITDATA *data;
 if (mac->bReceivedCTS)
 {
        // We've received the CTS. Send the data frame.
```

```
        data=mac->maccopydata;
  mac->frametype = DATA;
  mac->data_pos = 0;
  mac->data_size = data->datalen;
  mac->part = header;
  mac->header_pos = 0;
  mac->header_size = HEADER_SIZE;
  mac->duration = fduration (mac->frametype, mac->data_size);
  mac->uHeader.sHeader.duration=mac->duration;
        mac->uHeader.sHeader.FrameControl.Type=2;
        mac->uHeader.sHeader.FrameControl.SubType=0;
  mac->data = data->data;
  snprintf(mac->message, (size_t)MSG_MAX_SZ,
          "Got data unit of length %d from LLC.", mac->data_size);
  mac->setup->log(id, mac->message, lp_low);
  TXVECTOR* txv = create_TXVECTOR(mac->data_size + HEADER_SIZE);
  mac->BackOffTimer = 0;
  snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission Data start.");
  mac->setup->log(id, mac->message, lp_low);
  if (!TestChannelIdleAndBackOff(mac,SIFS))
  {
        // We were unable to send the DATA frame. Tell it to the LLC.
          status.tstat=undeliv_time;
          status.saddr=mac->uHeader.sHeader.Address1;
          status.daddr=mac->uHeader.sHeader.Address2;
          status.prov_prio=data->prio;
          status.prov_sclass=data->sclass;
          mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
   }
    else
   {
        mac->ACKfailCnt=0;
        mac->bReceivedACK=0;
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"Data Transmission start");
        mac->setup->log(id, mac->message, lp_low);
        mac->setup->PHY_TXSTART_request(id,txv);
   }
 }
 else
 {
          // We haven't received the CTS. Tell the LLC about the error.
          status.tstat=undeliv_time;  .Set the corresponding error from the standard.
          status.saddr=mac->uHeader.sHeader.Address1;
          status.daddr=mac->uHeader.sHeader.Address2;
          status.prov_prio=contention;
          status.prov_sclass=strictly_ordered;
          mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
 }
}
```

```
void ACKTimeout(MAC *mac,int iValue)
{
 const int id = mac->setup->staid;
 MA_UNITDATA_STATUS status;
 if (mac->bReceivedACK)
 {
        // We've received the ACK. Tell the LLC that everything was right.
        mac->ACKfailCnt=0;
        status.tstat=success;
        status.saddr=mac->uHeader.sHeader.Address1;
        status.daddr=mac->uHeader.sHeader.Address2;
        status.prov_prio=contention;
        status.prov_sclass=strictly_ordered;
        mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
 }
 else
 {
        // We haven't received the ACK.
        mac->ACKfailCnt++;
        if (mac->ACKfailCnt<MAXACKRETRY)
        {
                // Resend the frame.
              mac->data_pos = 0;
              mac->part = header;
              mac->header_pos = 0;
              TXVECTOR* txv = create_TXVECTOR(mac->data_size + HEADER_SIZE);
              mac->BackOffTimer = 0;
              snprintf(mac->message,(size_t)MSG_MAX_SZ,"Requesting transmission Data
start.");
              mac->setup->log(id, mac->message, lp_low);
              if (!TestChannelIdleAndBackOff(mac,SIFS))
              {
                      // We were unable to send the DATA frame. Tell it to the LLC.
                        status.tstat=undeliv_time;
                        status.saddr=mac->uHeader.sHeader.Address1;
                        status.daddr=mac->uHeader.sHeader.Address2;
                        status.prov_prio=data->prio;
                        status.prov_sclass=data->sclass;
                        mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
              }
                      else
                      {
                              mac->setup->PHY_TXSTART_request(id,txv);
                      }
        }
    else
    {
          // We've tried sending the data frame too many times. Discard it and
          // tell the LLC about it.
            status.tstat=undeliv_unack;
            status.saddr=mac->uHeader.sHeader.Address1;
```

```
            status.daddr=mac->uHeader.sHeader.Address2;
            status.prov_prio=contention;
            status.prov_sclass=strictly_ordered;
            mac->setup->MA_UNITDATA_STATUS_indicate(id,&status);
    }
  }
}

void FillSequence(MAC *mac)
{
  mac->SeqContNum = mac->SeqContNum + 1;
  mac->SeqContNum = mac->SeqContNum % 4096;  // When Counter>4096, back to 0
  // Set always fragment count to 0.
  mac->uHeader.header[22]=(mac->SeqContNum&0xF)<<4;
  mac->uHeader.header[23]=(mac->SeqContNum&0xFF0)>>4;
}

BOOL IsMediumBusy(MAC *mac)
{
        return (mac->NAV>0)||(mac->state==cca_busy);
}

BOOL TestChannelIdleAndBackOff(MAC *mac,int iTimer)
{
        BOOL bActive;
        int id = mac->setup->staid;
        // here we must wait iTimer nanoseconds and test all the time if the medium
        // gets active in this time. The simplest way is testing the NAV and force
        // the CCA_state function to set a flag that notifies this function about
        // activity.
        bActive=IsMediumBusy(mac);
        if (!bActive)
        {
                mac->bIdleTest=0;
               // mac->setup->registerTimeOut(id,iTimer,TimeoutIdle,mac);
                wait(iTimer);
        }
        // Esta el medio ocupado?
        if ((bActive)||(mac->bIdleTest!=0))
        return TestChannelIdleAndBackOffNoWait(mac);
        return 1;
}

BOOL TestChannelIdleAndBackOffNoWait(MAC *mac)
{
        int id = mac->setup->staid;
         // Medium is busy
        snprintf(mac->message,(size_t)MSG_MAX_SZ,"Medium is bussy, calling backoff.");
          mac->setup->log(id, mac->message, lp_low);
         // We have to wait
        return backoff (mac,id,mac->CW,mac->BackOffTimer);
```

```
}

void TimeoutIdle(MAC *mac,int iValue)
{
        if (mac->bIdleTest!=0)
         TestChannelIdleAndBackOffNoWait(mac);
}

BOOL backoff (MAC *mac,unsigned int a, unsigned int b, int c)
{
 unsigned uRetryCnt;
 unsigned uMaxRetryCnt; // This must be updated to be 4 or 7 depending
 if (RTSneed == 1)          // on the kind of frame transmitted (with or
 {                          // without RTS).
        uMaxRetryCnt=4;
 }
 else
{
         uMaxRetryCnt=7;
}
 uRetryCnt=0;
 mac->bIdleTest=1;
 while ((mac->bIdleTest)&&(uRetryCnt<uMaxRetryCnt))
 {
  if (c == 0)
        {
    c = GetRandomNumber (b);
        // if backofftimer is 0 we calculate another one
        }
  mac->bIdleTest=0;
  wait(c);
        if (b<CWMAX)
                b=b*2+1;
        else
                uRetryCnt++;
 }
 // If got out due to uRetryCnt<7, this means we cannot transmit (we did not get the
 // medium idle in any backoff waiting time). We must abort the transmission and tell
 // LLC that we were unable to send the frame.
 if (!mac->bIdleTest)
          return 1;
 return 0;
}

int GetRandomNumber(int iMaximum)
{
 return rand()%(iMaximum+1);
}
```

```
int fduration (Tframetype frametype, int lenght)
{
 int dur;
 if (frametype = RTS)  /* Este sera el caso RTS */
  {
   dur = (int)(5333.33+((((30+lenght)*8*1000)/54))+(3*SIFS));
   return (dur);
  }
 if (frametype = CTS)  /* Este sera el caso CTS */
  {
   dur = (int)(2962.96+((((30+lenght)*8*1000)/54))+(2*SIFS));
   return (dur);
  }
 if (frametype = DATA)  /* Este sera el caso DATA */
  {
   dur = (int)(1481.48+((((30+lenght)*8*1000)/54))+SIFS);
   return (dur);
  }
 if (frametype = ACK)  /* Este sera el caso ACK */
  {
   dur = (10*8)*1000/54; /* Esto viene a ser 1481.48ns */
   return (dur);
  }
 return 0;
}

void test(MAC* mac, int arg)
{
 int id = mac->setup->staid;
 snprintf(mac->message,(size_t)MSG_MAX_SZ,
          "Test routine activated, argument = %d", arg);
}

void wait(int iTime)
{
#ifdef WIN32
        Sleep(iTime/1000);
#else
        sleep(iTime/1000);
#endif
}
```

APPENDIX E: Files in relation with the implementation

Here are some of the files needed for understanding the MAC implementation code. These files can also be usefull for future modifications in the MAC implementation or for working with this code.

APPENDIX E1: Mactypes.c

```
#include "mac/mactypes.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

octet_t* createData(unsigned int len)
{
 if (len <= 0)
 {
   fprintf(stderr, "In %s: Buffer length must not be zero.\n",
__PRETTY_FUNCTION__);
   abort();
 }
 octet_t* dest = NULL;
 // Allocate a new data block with the required size.
 dest = (octet_t*)calloc((size_t)len, sizeof(octet_t));
 assert(dest != NULL);
 //  fprintf(stderr, "Allocated %d octets of data at address 0x%x\n",
 //        len, (unsigned int)dest);
 return dest;
}

octet_t* createCopyData(octet_t* src, unsigned int len)
{
 if (len <= 0)
 {
   fprintf(stderr, "In %s: Buffer length must not be zero.\n",
__PRETTY_FUNCTION__);
   abort();
 }
 octet_t* dest = NULL;
 // Allocate a new data block with the required size.
 dest = (octet_t*)calloc((size_t)len, sizeof(octet_t));
 assert(dest != NULL);
 //  fprintf(stderr, "Allocated %d octets of data at address 0x%x\n",
 //        len, (unsigned int)dest);
 // Copy the data to the allocated data packet.
```

```
 dest = memcpy(dest, src, len);
 return dest;
}

MA_UNITDATA*
create_MA_UNITDATA(macaddr_t sa, macaddr_t da, octet_t* dat, unsigned int len,
                   prio_t p, sclass_t s)
{
 MA_UNITDATA* ma_ud = (MA_UNITDATA*)malloc(sizeof(MA_UNITDATA));
 if (ma_ud == NULL) {
  fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
  abort();
  }
 ma_ud->saddr = sa;
 ma_ud->daddr = da;
 ma_ud->rinfo = 0;
 ma_ud->data = dat;
 ma_ud->datalen = len;
 ma_ud->rstat = SUCCESS;
 ma_ud->prio = p;
 ma_ud->sclass = s;
 return ma_ud;
}

MA_UNITDATA_STATUS*
create_MA_UNITDATA_STATUS(macaddr_t sa, macaddr_t da, tstat_t ts,
                          prio_t pp, sclass_t ps)
{
 MA_UNITDATA_STATUS* ma_uds = (MA_UNITDATA_STATUS*)
   malloc(sizeof(MA_UNITDATA_STATUS));
 if (ma_uds == NULL)
 {
  fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
  abort();
  }
 ma_uds->saddr = sa;
 ma_uds->daddr = da;
 ma_uds->tstat = ts;
 ma_uds->prov_prio = pp;
 ma_uds->prov_sclass = ps;
 return ma_uds;
}

MAC_SETUP*
create_MAC_SETUP(void)
{
```

```
 MAC_SETUP* mas =  (MAC_SETUP*)malloc(sizeof(MAC_SETUP));
 if (mas == NULL)
{
   fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
   abort();
 }
 return mas;
}

TXVECTOR*
create_TXVECTOR(unsigned int len)
{
 TXVECTOR* txv =  (TXVECTOR*)malloc(sizeof(TXVECTOR));
 if (txv == NULL)
{
   fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
   abort();
 }
 txv->rate = 54;
 txv->length = len;
 return txv;
}

RXVECTOR*
create_RXVECTOR(unsigned int len)
{
 RXVECTOR* rxv =  (RXVECTOR*)malloc(sizeof(RXVECTOR));
 if (rxv == NULL)
{
   fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
   abort();
 }
 rxv->rate = 54;
 rxv->length = len;
 return rxv;
}

void destroyData(octet_t* data)
{
  free(data);
}

void destroy_MA_UNITDATA(MA_UNITDATA* mad)
{
  free(mad);
}
```

```
void destroy_MA_UNITDATA_STATUS(MA_UNITDATA_STATUS* mads)
{
 free(mads);
}

void destroy_MAC_SETUP(MAC_SETUP* mas)
{
 free(mas);
}

void destroy_TXVECTOR(TXVECTOR* txv)
{
 free(txv);
}

void destroy_RXVECTOR(RXVECTOR* rxv)
{
 free(rxv);
}

void print_MA_UNITDATA(MA_UNITDATA* mad)
{
 assert(mad != NULL);
 fprintf(stderr,"Printing MA_UNITDATA struct at address %x:\n", mad);
 fprintf(stderr,"Data length is %u bytes.\n", mad->datalen);
 fprintf(stderr,"Data is located at address %x and is:\n", mad->data);
 assert(mad->data != NULL);
}

void macaddrSetAddr(macaddr_t* ma, unsigned long long address)
{
 ma->addr[0] = (octet_t)((address >> 0) & 0xFF);
 ma->addr[1] = (octet_t)((address >> 8) & 0xFF);
 ma->addr[2] = (octet_t)((address >> 16) & 0xFF);
 ma->addr[3] = (octet_t)((address >> 24) & 0xFF);
 ma->addr[4] = (octet_t)((address >> 32) & 0xFF);
 ma->addr[5] = (octet_t)((address >> 38) & 0xFF);
}

short macaddrCompare(macaddr_t* a, macaddr_t* b)
{
 return a->addr[0] == b->addr[0];
   a->addr[1] == b->addr[1];
   a->addr[2] == b->addr[2];
   a->addr[3] == b->addr[3];
```

```
   a->addr[4] == b->addr[4];
   a->addr[5] == b->addr[5];
}

char* macaddrToString(macaddr_t* a)
{
  size_t s = 18;
  char* str = malloc(s);
  snprintf(str, s, "%02hhX:%02hhX:%02hhX:%02hhX:%02hhX:%02hhX",(int)a-
>addr[0],
            (int)a->addr[1], (int)a->addr[2], (int)a->addr[3],
            (int)a->addr[4], (int)a->addr[5]);
  return str;
}
```

APPENDIX E2: Mactypes.h

```
#ifndef MACTYPES_H_HEADER_INCLUDED
#define MACTYPES_H_HEADER_INCLUDED


#ifndef __cplusplus

/* Code for C */

typedef struct macaddr_t macaddr_t;
typedef struct MAC MAC;
typedef MAC* mac_handle_t;
typedef struct MA_UNITDATA MA_UNITDATA;
typedef struct MA_UNITDATA_STATUS MA_UNITDATA_STATUS;
typedef struct MAC_SETUP MAC_SETUP;
typedef struct TXVECTOR TXVECTOR;
typedef struct RXVECTOR RXVECTOR;

typedef enum log_level_t log_level_t;
typedef enum log_prio_t log_prio_t;
typedef enum command_t command_t;
typedef enum prio_t prio_t;
typedef enum sclass_t sclass_t;
typedef enum tstat_t tstat_t;
typedef enum CCASTATE CCASTATE;
typedef enum RXERROR RXERROR;

/* These are defined in types.hh for c++ */
```

```
typedef long long mtime_t;

typedef unsigned char octet_t;

typedef octet_t phy_data_t;

enum log_level_t {log_none=4, log_low=3, log_medium=2, log_high=1, log_debug=0};

enum log_prio_t {lp_debug=0, lp_low=1, lp_medium=2, lp_high=3, lp_alert=4};

enum command_t {cmd_run, cmd_step, cmd_load, cmd_quit, cmd_add,
                cmd_restart};

#define SUCCESS     1
#define NANOSECOND  1ULL
#define MICROSECOND 1000ULL
#define MILLISECOND 1000000ULL
#define SECOND      1000000000ULL
#define MINUTE      60000000000ULL
#define HOUR        3600000000000ULL

#else /* __cplusplus */

const int SUCCESS = 1;

struct MAC;
typedef MAC* mac_handle_t;

#include "types.hh"

#endif /* __cplusplus */


enum prio_t {contention, contention_free};

enum sclass_t {reorderable_multicast, strictly_ordered};

enum tstat_t {success, undeliv_unack, excess_len, non_null_src_rout,
              prio_not_supp, sclass_not_supp, prio_not_av,
              sclass_not_av, undeliv_time, undeliv_no_bss,
              undeliv_null_encrypt};

enum CCASTATE {cca_busy, cca_idle};

enum RXERROR {no_error, format_violation, carrier_lost,
```

```
                unsupported_rate};

struct macaddr_t
{
 octet_t addr[6];
};

struct MA_UNITDATA
{
 macaddr_t saddr;
 macaddr_t daddr;
 int rinfo;
 octet_t* data;
 unsigned int datalen;
 int rstat;
 prio_t prio;
 sclass_t sclass;
 unsigned int duration;
 int frametype;
}; /* MA_UNITDATA */

struct MA_UNITDATA_STATUS
{
 macaddr_t saddr;
 macaddr_t daddr;
 tstat_t tstat;
 prio_t prov_prio;
 sclass_t prov_sclass;
};

struct TXVECTOR
{
 int rate;
 unsigned int length;
};

struct RXVECTOR
{
 int rate;
 unsigned int length;
};

struct MAC_SETUP
{
 int staid;
 macaddr_t macaddr;
```

```
  void (*MA_UNITDATA_indicate)(int, MA_UNITDATA*);
  void (*MA_UNITDATA_STATUS_indicate)(int, MA_UNITDATA_STATUS*);
  void (*PHY_DATA_request)(int, octet_t);
  void (*PHY_TXSTART_request)(int, TXVECTOR*);
  void (*PHY_TXEND_request)(int);
  void (*PHY_CCARESET_request)(int);
  void (*registerTimeOut)(int, mtime_t, void (*)(mac_handle_t, int), int);
  void (*log)(int, char*, log_prio_t);
  mtime_t (*getTime)(int);
};

#ifdef __cplusplus
extern "C"
{
#endif
octet_t*
createData(unsigned int len);

octet_t*
createCopyData(octet_t* src, unsigned int len);

MA_UNITDATA*
create_MA_UNITDATA(macaddr_t sa, macaddr_t da, octet_t* dat, unsigned int len,
                   prio_t p, sclass_t s,unsigned int duration, int frametype);

MA_UNITDATA_STATUS*
create_MA_UNITDATA_STATUS(macaddr_t sa, macaddr_t da, tstat_t ts,
                          prio_t pp, sclass_t ps);

MAC_SETUP*
create_MAC_SETUP(void);

TXVECTOR*
create_TXVECTOR(unsigned int len);

RXVECTOR*
create_RXVECTOR(unsigned int len);

void
destroyData(octet_t* data);

/*! \brief Deallocate a MA_UNITDATA object.
 * Deallocate a MA_UNITDATA object. Note that the data field should be handled
 * separately, since it isn't created by the create_MA_UNITDATA function.
 *
 */
```

```
void destroy_MA_UNITDATA(MA_UNITDATA* mad);

void destroy_MA_UNITDATA_STATUS(MA_UNITDATA_STATUS* mads);

void destroy_MAC_SETUP(MAC_SETUP* mas);

void destroy_TXVECTOR(TXVECTOR* txv);

void destroy_RXVECTOR(RXVECTOR* rxv);

void print_MA_UNITDATA(MA_UNITDATA* mad);

void macaddrSetAddr(macaddr_t* ma, unsigned long long address);

short macaddrCompare(macaddr_t* a, macaddr_t* b);

char* macaddrToString(macaddr_t* a);

#ifdef __cplusplus
}
#endif

#endif /* MACTYPES_H_HEADER_INCLUDED */
```

APPENDIX E3: MAC.C

```
#include <stdlib.h>
#include <stdio.h>
#include "mactypes.h"

struct MAC {
  // Save the setup data here
  MAC_SETUP* setup;
  octet_t* data;
  int data_pos;
  int data_size;
};

/* ================================================================
 * Function declarations (prototypes)
 * ================================================================ */

MAC* create_MAC(void);
```

```
void initialise_MAC(MAC* mac, MAC_SETUP* macs);

void destroy_MAC(MAC* mac);

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data);

void PHY_DATA_confirm(MAC* mac);

void PHY_TXSTART_confirm(MAC* mac);

void PHY_TXEND_confirm(MAC* mac);

void PHY_CCARESET_confirm(MAC* mac);

void PHY_DATA_indicate(MAC* mac, octet_t data);

void PHY_CCA_indicate(MAC* mac, CCASTATE state);

void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv);

void PHY_RXEND_indicate(MAC* mac, RXERROR rxe);

/* ============================================================== */

MAC* create_MAC(void)
{
 MAC* mac = (MAC*)malloc(sizeof(MAC));
 if (mac == NULL) {
  fprintf(stderr, "Out of memory in %s", __PRETTY_FUNCTION__);
  abort();
  }
 mac->setup = NULL;
 mac->data = NULL;
 mac->data_size = 0;
 mac->data_pos = 0;
 printf("MAC created!\n");
 return mac;
}

void initialise_MAC(MAC* mac, MAC_SETUP* macs)
{
 if (mac->setup != NULL) {
   destroy_MAC_SETUP(mac->setup);
  }
 mac->setup = macs;
}
```

```
void destroy_MAC(MAC* mac)
{
 if (mac != NULL) {
   destroy_MAC_SETUP(mac->setup);
   free(mac->data);
   free(mac);
  }
}

void MA_UNITDATA_request(MAC* mac, MA_UNITDATA* data)
{

}

void PHY_DATA_confirm(MAC* mac)
{
}

void PHY_TXSTART_confirm(MAC* mac)
{
}

void PHY_TXEND_confirm(MAC* mac)
{
}

void PHY_CCARESET_confirm(MAC* mac)
{
}

void PHY_DATA_indicate(MAC* mac, octet_t data)
{
}

void PHY_CCA_indicate(MAC* mac, CCASTATE state)
{
  // Receive indication of PHY state change
}

void PHY_RXSTART_indicate(MAC* mac, RXVECTOR* rxv)
{
}
```

```
void PHY_RXEND_indicate(MAC* mac, RXERROR rxe)
{
  // End receiving transmission
}
```

Copyrigth

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/