Master Thesis

Analysis and Development of TDMA Based Communication Scheme for Car-to-Car and Car-to-Infrastructure Communication Based on IEEE802.11p and IEEE1609 WAVE Standards

by

Cristina Cocho

A thesis submitted in the Institut für Nachrichtentechnik und Hochfrequenztechnik at the Technishen Universität Wien

Wien, March 2009

Thesis performed in department of Programm und Systementwicklung of Siemens Österreich

In collaboration with the Institut für Nachrichtentechnik und Hochfrequenztechnik

from Technische Universität Wien

And supervised by the Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad Politécnica de Madrid.

- Director: Univ.Prof. Dipl.-Ing. Dr.-Ing. Christoph MECKLENBRÄUKER (TU Wien)

- Tutor: Dipl.-Ing. Dr. Alexander PAIER (TU Wien)

- Rapporteur: Univ.Prof. Dipl.-Ing. Dr.-Ing. Alberto Almendra (ETSIT from Universidad Politécnica of Madrid)

Abstract

Safety critical Intelligent Transportation Systems (ITS) applications provide information to vehicles to avoid potentially dangerous traffic situations or to reduce the seriousness of an accident. This information, when received well in advance, provides an early warning to the driver and becomes increasingly time-critical as the vehicle approaches the site of an incident or potential accident. It can be seen, therefore, that these communications must be reliable, have a high success rate and not suffer from excessive latency.

In Europe it was concluded, after a study of the spectrum requirements in the 5.9 GHz band, carried out by the Commission of European Post and Telecommunications (CEPT) that at least 30 MHz were necessary for "safety related applications" in the frequency range 5875-5905 MHz. Within this spectrum a dedicated allocation of bandwidth usage has been proposed by the European Telecommunications Standards Institute, ETSI. For high usage of bandwidth the preliminary standards IEEE 802.11p and IEEE 1609.4 have proposed adjacent 10 MHz channels which may cause interference using low cost WLAN Chipsets.

The goal of this diploma thesis is to analyse and develop an alternative scheme based on Time Division Multiplex Access (TDMA) technology to avoid these channel interferences. The Network Simulator (version 2.33) and the IEEE family standards 802.11 and 1609 will be the main tools used to carry out the diploma.

Firstly the TDMA based protocol will be defined theoretically and later introduced in the source code of the Network Simulator. Once the protocol is debugged, some test environments (written in Tool Command Language code) will be set up to obtain different trace files that lately will be used to obtain graphical results by using Perl scripts. Finally those results will be used to compare the actual Frequency Division Multiplex Access (FDMA) based protocol with our TDMA based protocol developed.

Acknowledgements

This Diploma Master Thesis would not have being done without the help and support of some people. First I would like to thank the supervision of the thesis director at the Vienna University of Technology, Prof. Christoph Mecklenbräuker and the tutor Alexander Paier. Thank you for being extremely patient with me even when I was finishing the diploma in Madrid. You always answer me all the questions and try to make my work easier each day.

I also would like to offer my gratitude to the people from the PSE CVD CON department of Siemens Österreich place where I mainly did my diploma. Especially thanks to Herbert Füreder and the rest of people I was working with for explaining me all the knowledge necessary to begin the diploma.

Thanks also to Manuel Zaera, an Erasmus student and friend, who really help me with the work related to the Network Simulator, your information given and your suggestions were special important at the beginning and at the end of the diploma.

Special thanks to my boyfriend for always supporting me; even in the moments when I was not enthusiastic about the work done. I am also really grateful to my family who always understood my problems and was comprehensive with me. To my parents, Esperanza and Lucio, for giving me support and show always a big interest in my diploma and to my sister, Blanca, for her advices.

Contents

List of Figures

List of Tables

Acronyms

ACK	**ACK**nowledgement
BPSK	**B**inary **P**hase **S**hift **K**eying
C2C	**C**ar **T**o **C**ar (communications)
C2I	**C**ar **T**o **I**nfrastructure
C2X	**C2I + C2C**
CCH	**C**ontrol **Ch**annel
CEPT	**C**ommission of **E**uropean **P**ost and **T**elecommunications
CSMA/CA	**C**arrier **S**ense **M**edium **A**ccess/ **C**ollision **A**voidance
CTS	**C**lear **T**o **S**end
CW	**C**ontention **W**indow
DCF	**D**istributed **C**oordination **F**unction
DIFS	**DCF I**nterframe **S**pace
DiffServ	**Diff**erentiated **Serv**ices
DSDV	**D**estination **S**equence **D**istance **V**ector
EDCA	**E**nhanced **D**istributed **C**hannel **A**ccess
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
FCC	**F**ederal **Co**mission **C**ommunication
FCS	**F**rame **C**heck **S**equence
FDMA	**F**requency **D**ivision **M**ultiplex **A**ccess
FTP	**F**ile **T**ransfer **P**rotocol
GloMoSim	**Glo**bal **Mo**bile Information System **Sim**ulation Library
IEEE	**I**nstiture of **E**lectrical and **E**lectronics **E**ngineers, Inc.
IP	**I**nternet **P**rotocol
ITS	**I**ntelligent **T**ransportation **S**ystems
LAN	**L**ocal **A**rea **N**etwork
LLC	**L**ogic **L**ink **C**ontrol
MAC	**M**edium **A**ccess **C**ontrol
MIB	**Ma**nagement **I**nformation **B**ase
MLME	**MAC L**ayer **M**anagement **E**ntity
MSDU	**MAC S**ervice **D**ata **U**nit
MPDU	**MAC P**rotocol **D**ata **U**nit
NAM	**N**etwork **Ani**Mator (Network Simulator visualization tool)
NAV	**N**etwork **A**llocation **V**ector
NDBPS	**N**umber of **D**ata **B**its **P**er **OFDM S**ymbol
NS	**N**etwork **S**imulator
OBU	**O**n **B**oard **U**nit
OFDM	**O**rthogonal **F**requency **D**ivision **M**ultiplexing
OTcl	**O**bject **T**ool **C**ommand **L**anguage
PBC	**P**eriodic **B**roadcast **P**rotocol
PERL	**P**ractical **E**xtraction and **R**eport **L**anguage
PLME	**P**hysical **L**ayer **M**anagement **E**ntity
PHY	**PHY**sical Layer
QAM	**Q**uadrature **A**mplitude **M**odulation
QPSK	**Q**uadrature **P**hase **S**hift **K**eying

RED	**R**andom **E**arly **D**etection
RF	**R**adio **F**requency
RSU	**R**oad **S**ide **U**nit
RTS	**R**eady **T**o **S**end
SDMA	**S**pace **D**ivision **M**ultiple **A**ccess
SCH	**S**ervice **CH**annel
SIFS	**S**hort **I**nter**f**rame **S**pacing
STDMA	**S**patial reuse of **TDMA**
SUMO	**S**imulation of **U**rban **MO**bility
Tcl	**T**ool **C**ommand **L**anguage
TCP	**T**ransfer **C**ontrol **P**rotocol
TDMA	**T**ime **D**ivision **M**ultiplex **A**ccess
TraNS	**TRA**ffic and **N**etwork **S**imulation environment
UDP	**U**ser **D**ata **P**rotocol
UP	**U**ser **P**riority
UTC	**C**oordinated **U**niversal **T**ime
U.S	**U**nited **S**tates
VANET	**V**ehicular **A**d hoc **NET**work
VISSIM	Geman acronym of **T**raffic **I**n **T**owns **SIM**ulator
WAVE	**W**ireless **A**ccess in **V**ehicular **E**nvironments
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WME	**WAVE M**anagement **E**ntity
WSA	**WAVE S**ervice **A**dvertisement
WSM	**W**ave **S**hort **M**essage
WSMP	**W**ave **S**hort **M**essage **P**rotocol

To my parents, Esperanza and Lucio,
and my sister Blanca

1 Introduction

Nowadays there is an increasing interest in wireless communications standards for Intelligent Transportation Systems (ITS). Those standards are mainly defined to be used in traffic safety and non-safety applications. Safety applications provide drivers information about critical situations in advance (a critical situation could be the car in front of you suddenly stops) and require strict reliability and delay. Non-safety applications improve driving comfort and usually are more bandwidth sensitive. Examples of those non-safety applications are on board internet access and driving through payment.

Both types of applications are used in Car to Car (C2C) and Car to Infrastructure (C2I) communications which in general receive the name of C2X communications. C2X communications are defined by the IEEE 1609 and IEEE 802.11p standards. Those standards establish an IEEE 802.11 Wireless Local Area Network (WLAN) communication system, which is called Wireless Access in Vehicular Environments (WAVE). The system diagram of WAVE communications is shown in Figure 1.1:

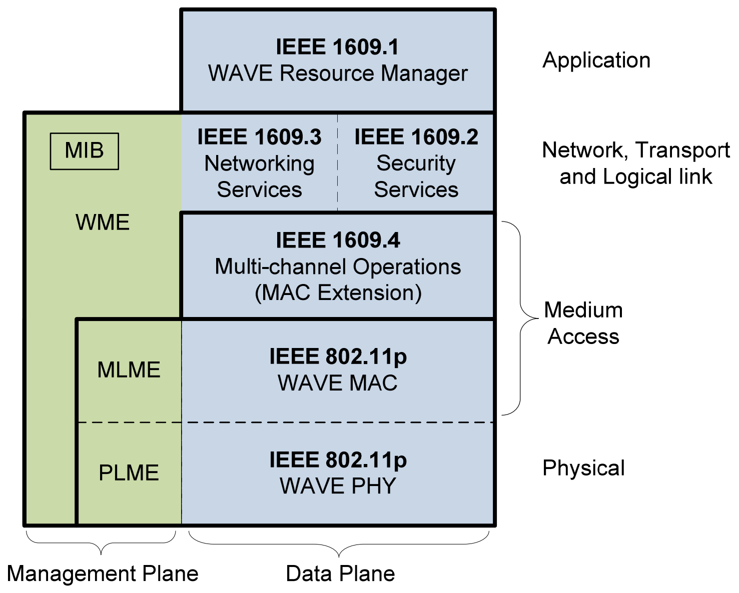


Figure 1.1: Diagram of the protocol stack in a WAVE system. Figure based in [1], [2].

As we can see in Figure 1.1 the Physical layer (PHY) and the basic Medium Access Control (MAC) layer are specified by the standard IEEE 802.11p while the upper layers are defined by the IEEE 1609 standard family. In this diploma our work will be based in MAC layer so we will be really interested in using the standards IEEE 1609.4 and IEEE 802.11p.

The IEEE 1609 family standards define two types of communication channels in WAVE systems to support safety and non-safety applications. On one hand we have the Control Channel (CCH) which is used to transmit WAVE Short Messages (WSMs) and announce WAVE services [3] on the other hand we have the Service Channel (SCH) which is used for application interactions/transmissions. Any WAVE system will support one CCH and one or more SCH. The existence of more than one SCH will depend on the system requirements and the available bandwidth.

The bandwidth allocated to ITS wireless communications is nowadays 75 MHz in the 5.850-5.925 GHz frequency band, although its usage depends if we are in United States (U.S) or in Europe. In U.S the bandwidth (approved by Federal Communication Commission, FCC, in 1999) is divided into seven channels, each with 10 MHz. Their actual frequency allocation is shown in Figure 1.2:

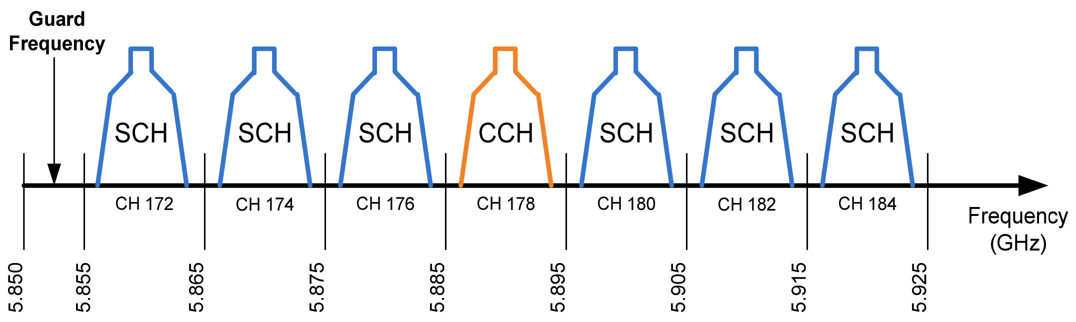


Figure 1.2: Distribution of the 75 MHz bandwidth in U.S for ITS wireless communications. Figure based in [2].

In Europe, the 75 MHz bandwidth is used in a different way. The European Telecommunications Standards Institute (ETSI) defines the frequency band 5.855-5.875 GHz for non-safety applications and the frequency band 5.875-5.925 GHz for safety applications. The usage of the bandwidth for safety applications will be done in two phases: in the first phase only the band from 5.875-5.905 GHz (bandwidth used nowadays) will be used and in the second phase this bandwidth will be extended to 5.925 GHz. This means actually there are only 30 MHz available for WAVE communications (instead of the 70 MHz used in U.S) as we can see in Figure 1.3:

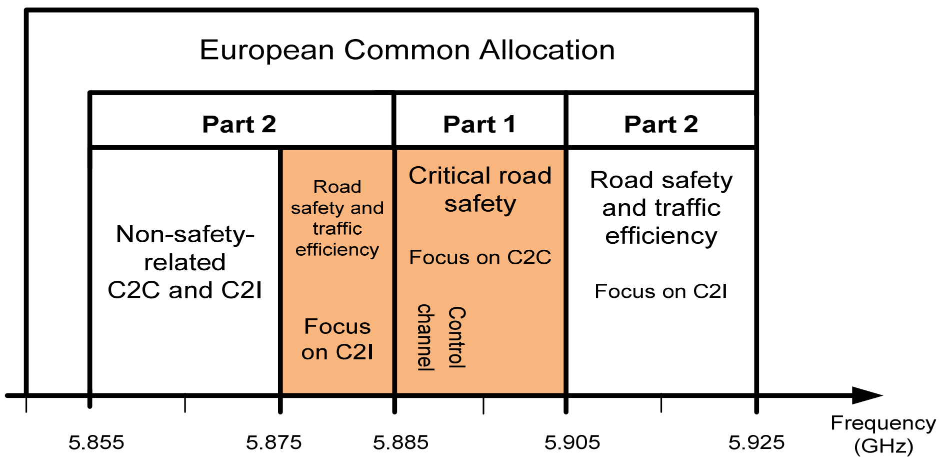


Figure 1.3: European spectrum allocation for ITS wireless communications. Figure based in [1].

There are different ways of using this 30 MHz depending if it is required a robust system or small channel interference. There are documents where the usage of different bandwidth for each channel is analysed [4] although nowadays the most common option is to use channels of 10 MHz.

In a WAVE system there are also two types of devices: the Roadside Units (RSUs) and the Onboard Units (OBUs). An RSU is a WAVE device that operates at a fixed position (usually along the road transport network) that supports communication and data exchange with OBUs. An OBU is a mobile or portable WAVE device that supports information exchange with RSUs and other OBUs.

Both WAVE devices make use of the CCH and SCHs communication channels to get information about safety and non-safety applications. Usually the process is the following: a WAVE device always begins monitoring the CCH during specific intervals of time (known as control channel intervals). During this time the device can receive two types of information: safety (or private service advertisements) and non-safety information. Non-safety information is basically information about the services which are going to be offered by other WAVE devices during the following SCH interval. There are two ways of receiving non-safety information during the CCH interval: through a WAVE service advertisement (WSA) sent by another WAVE device or through a WAVE announcement frames transmitted by our WAVE device (see page 5 of [3]).

After the CCH interval always comes a SCH interval where different services are offered. The WAVE device will monitor the SCH channel if it is interested in one application offered in this interval, otherwise the device will continue monitoring the CCH. A schematic of this process is shown in Figure 1.4.

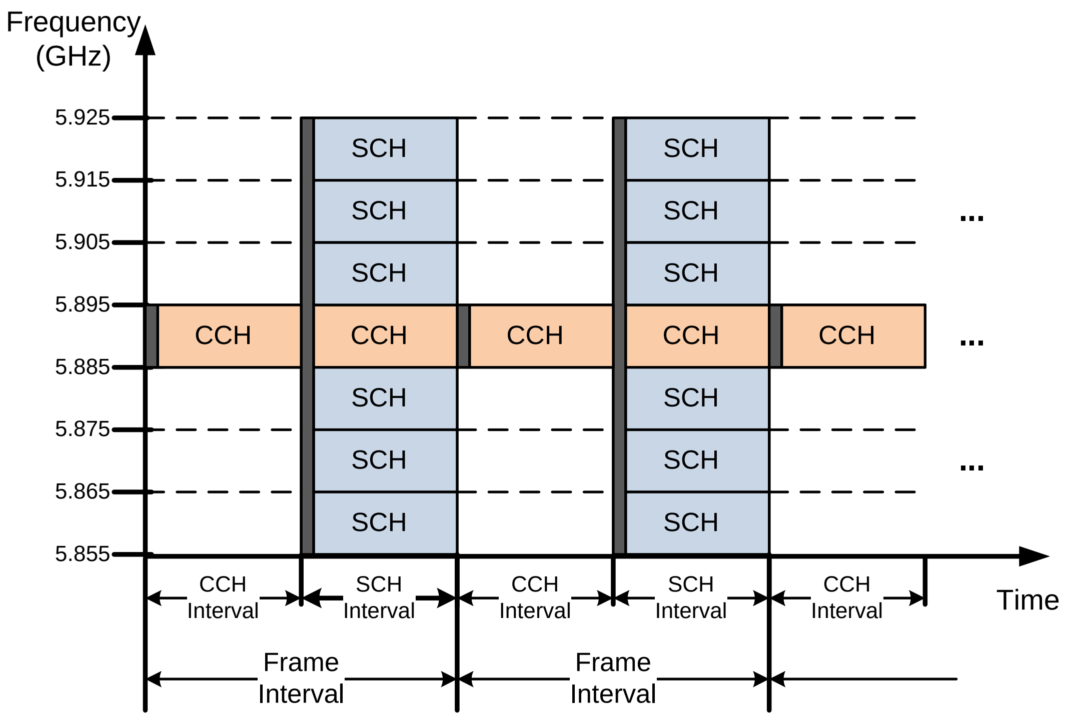


Figure 1.4: Distribution of service and control channels in time and frequency. Figure based in [2].

This figure illustrates how both types of channels are used in the time domain in U.S. In case of Europe we must keep in mind that only two SCH are offered nowadays.

Each of the SCH is also offered during an interval (as we can see in the in figure 1.4), which means it is necessary to suspend the data transactions on a SCH when CCH interval begins in case we have a single transceiver (a single channel device that can perform exchanges on only one Radio Frequency, RF, channel at a time). If a WAVE device has not consumed all the data during an SCH interval the process will be resumed when CCH monitoring is no longer required. To avoid losing packets it is important that any WAVE device supports buffering data packets while monitoring the CCH.

But not only buffering is necessary. Another important point is synchronization. Synchronization is the procedure by which a device adopts the time reference of another source of time. Synchronization means not only that WAVE devices must be synchronized to each other but also they must know when it is permissible to cease monitoring the CCH. An absolute external time reference, the Coordinated Universal Time (UTC), is used to define CCH and SCH intervals uniquely. There is also the possibility of using dual transceiver which allows receiving simultaneously a CCH and a SCH [6].

Both CCH and SCHs are sent in different frequencies so if a client is interested in receiving information about a specific service it will have to change the tuned frequency at the beginning of the service channel interval, or time slot, in case a single transceiver is used. This means a FDMA (Frequency Division Multiplex Access) technology is being used to handle different channels. With FDMA it is possible to transmit more than one communication channel at the same time allowing any user to receive the channel it is interested in. In case of WAVE communications, FDMA allows different users to make use of different services (transmitted in different SCHs) at the same time, as we can see in Figure 1.4.

But FDMA technology has also some disadvantages; one of them is channel interference produced by dispersion of the signal transmitted which increases the packet error rate (Figure 1.5). Having large packet error rates is a serious problem especially when transmitting safety information. This channel interference is not only produced by adjacent channels but also between non-adjacent channels although in the former case the interference is higher.

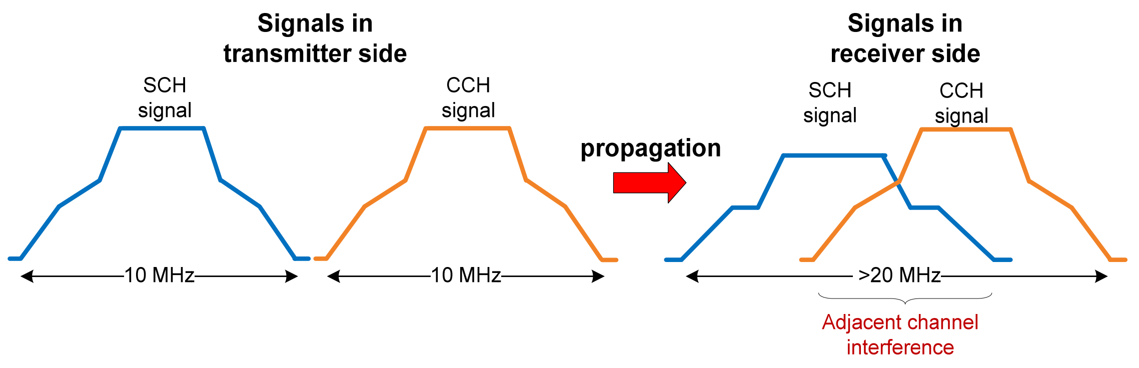


Figure 1.5: Adjacent channel interference between control and service channel.

To reduce those interference and hence to improve the robustness of the system there are solutions based in changing the bandwidth of each channel (using channels of 5 MHz or 20 MHz) or changing the position of the channels (changing its frequency) as it is explained in [4]. This channel interference motivated our diploma. The main reason why we decided to study the usage of Time Divison Multiplex Access (TDMA) technology in C2X communications was to see if TDMA is a good option to avoid channel interference.

What does it mean using TDMA instead of FDMA in a WAVE system? The idea is the following: as mentioned before we are interested in reducing or avoiding channel interference. This channel interference is produced by the existence of more than one communication channel at the same time (parallel channels), which means we need to use FDMA to access to different channels. Obviously we will not have channel interference if we use the available bandwidth (30 MHz in Europe) only to send one channel at each time. But if we are only able to send one channel at each time we need another multiplexing technique to offer more than one channel; this technique is TDMA.

In Figure 1.6 we can see the main differences between using FDMA or TDMA:

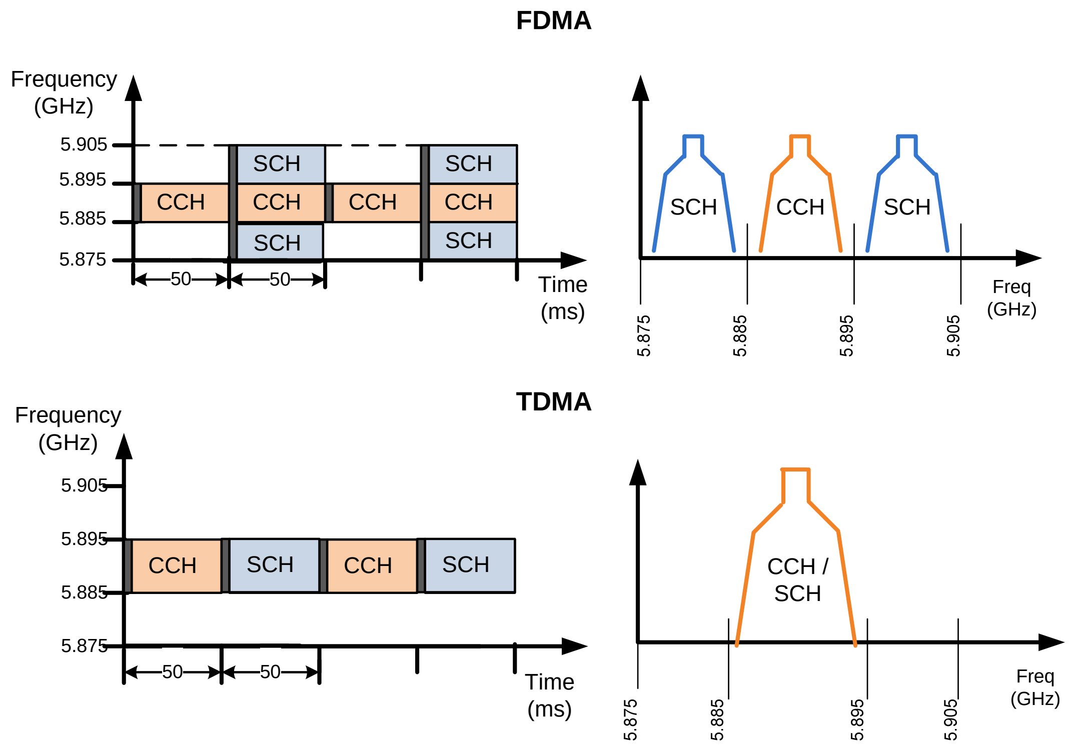


Figure 1.6: Comparison of TDMA and FDMA techniques in WAVE communications. Figure based in [2].

Although, when using TDMA all the bandwidth is utilized to send one channel this does not mean the channel will have a bandwidth of 30 MHz. Basically we will analyse what happens when multiplexing 10 MHz channels because we are only interested in changing the multiplexing technique, but not the devices and as it is said in paragraph 3.3 of [4], implementing 30 MHz channels requires to make use of new filters. Also we must keep in mind that after implementing the TDMA protocol we would like to compare it with the actual FDMA based implementation which makes use of 10 MHz communication channels. This is the main reason why we will work with 10 MHz channels.

TDMA is a scheduled-protocol [7] (or conflict-free protocol [8]). Scheduled-based protocols are highly sensitive to the network topology, which constitutes one of their main disadvantages, because usually any change in network topology will require a reconfiguration of the TDMA frame. These changes are necessary to reduce the latency produced when the assignation of the time slots to the users is wrong in distributed systems (see definition of distributed systems in [8]). The advantage of scheduled based protocols is the reduced number of collisions.

There is another category of vehicular MAC protocols: contention-based protocols [7], [8]. Contention-based protocols have the advantage that they are not sensitive to mobility and topology changes (characteristic of VANETS), the disadvantage is the unbounded delay because of the random access to the medium. An example of contention-based MAC protocol is Carrien Sense Multiplex Access with Collision Avoidance (CSMA/CA).

When trying to define our protocol we found a lot of documents which use TDMA based techniques in vehicular communications. Most of them establish distributed systems and try to define algorithms to improve the allocation of each user in a different slot of the frame.

One really interesting paper for us is [7], [9], because it defines an algorithm which allows the cars to self-configure the TDMA frame to reduce the delay produced when sending frames in distributed system. Another interesting protocol is the Spatial reuse of TDMA (STDMA, [10]), which can be considered as an extension of TDMA to increase the capacity of the protocol, in order to adapt the use of the time slots to the changes in the network topology.

Other protocols make use of the advantages of different multiplexing techniques like Z-MAC [11] and D-RAND [12] where the protocol defined acts as CSMA under low contention and as TDMA under high contention.

There are some wireless sensor protocols whose ideas can be also applied in WAVE communications. From all of them we can point out S-MAC protocol [13], which introduces the idea of using sleep intervals to reduce the power consumption (caused by idle listening). An improvement of S-MAC protocol is D-MAC [14] where the duration of the sleep intervals is variable to adapt the system to the traffic load reducing packet delivery latency and TDMA-W [15], where Transmit/Send and Wakeup slots are defined.

Finally it is also possible to find documentation about other multiplexing techniques used in Vehicular Ad Hoc Networks (VANETS) like for example Space Division Multiple Access (SDMA, [16]) in which the road is “divided” in space divisions and each vehicle is allowed to access the channel only at the time slot corresponding to the space division in which it is allocated.

2 Tools Explanation

In this chapter we are going to explain the main characteristics about the tools used in the diploma. These tools can be divided in two groups: the simulation tool and the IEEE standards. Once we knew the reason why we were interested in studying TDMA technology in C2X communications we needed to define how we were going to do this study. Basically, once we have defined our technical aspects of the MAC protocol we must find the best simulator to do that.

The idea is: we want to set up a simulator environment and modify it to get the desired behaviour. Usually a simulator environment consists of two logical elements [17] and [18]: a traffic simulator which is responsible for generating the mobility of vehicles and a network simulator which is, in our case, dedicated to represent the functionality of a real wireless network (for example a VANET) with all its complex effects of mobile communications. The traffic simulator gives periodically the positions of the vehicles that participate in the network to the network simulator in order to have the current connectivity pattern available. Sometimes a third component is defined, the application, which is in charge of controlling the whole simulation environment. Although this application can be implemented as an additional module, but commonly it is included in the network simulator. That is the reason why it is usually said that the simulator environment is defined by two components.

There are several traffic simulators available. One example is SUMO (Simulation of Urban Mobility) [19] a microscopic, space continuous and time discrete road traffic simulator package (further details such as the definition of microscopic and space continuous, can be obtained from [20]). Other traffic simulators are VISSIM (German acronym for Traffic In Town SIMulation) [21] and CARISMA a traffic simulator developed by BMW.

If we pay attention to the network simulators which better fit in our purpose we can point out the Network Simulator (whose characteristics we will explain later), the GloMoSim (Global Mobile Information System Simulation library) [22] or the NCTUns (Network Simulator and Emulator) [23] which is an open-source software running on Linux whose 5.0 release [24] has a complete implementation of the IEEE 802.11p and 1609 standards defined for wireless vehicular networks. The OMNet++ [25] is a discrete event simulation environment, whose primary application area is the simulation of communication networks, but that is successfully used in other areas like the simulation of complex IT systems and queuing networks. This simulator is also open source available. Sometimes it is also necessary to use an intermediate between traffic and network simulator to obtain more realistic simulations, this is the case of TraNS (TRAffic and Network Simulation Environment) [26] a tool that nowadays is used to link the traffic simulator SUMO and the network simulator ns2 [27].

In our case we were more interested in defining the application and MAC layer, than having a realistic movement pattern of the nodes, that is why we decided to focus on network simulators. From all of them we decided to make use of the Network Simulator (NS), because it is nowadays the most widely used in wireless simulations.

NS [28] is an object oriented simulator developed at UC Berkeley that simulates variety of Internet Protocol (IP) networks. It implements network protocols such as Transfer Control Protocol (TCP) and User Data Protocol (UDP), traffic source behaviour such as File Tranfer Protocol (FTP) and Telnet, router queue management mechanism such as Drop Tail and RED (Random Early Detection) and more. NS also implements multicasting and some of the MAC layer protocols for Local Area Networks (LAN) simulations. The NS is currently based on two languages [29]: an object oriented simulator, written in C++, and an OTcl (an object oriented extension of Tool Command Language, Tcl) interpreter, used to execute user's command scripts. Due to the usage of two programming languages, the simulator supports two class hierarchies: the compiled C++ hierarchy and the interpreted OTcl one, with one to one correspondence between them.

The reason why two languages are used is to fulfil different requirements (page 19 in [30]): on the one hand the compiled C++ hierarchy allows us to achieve efficiency in the simulation and faster execution time when defining and working with protocols. This is useful to reduce processing time when necessary. On the other hand, sometimes we are not interested in having a fast execution of the code but in being able to change parameters or configurations and quickly exploring a number of scenarios. In these cases, where the iteration time (time destined to change the model defined and re-run it) is more important, the interpreted OTcl hierarchy is used.

Usually the user defines an OTcl script which includes information about a particular network topology, the specific protocols and applications that he wants to simulate (whose behaviour is already defined in the compiled hierarchy) and the form of the output from the simulator. This OTcl script contains simulator objects which are instantiated within the OTcl interpreter and mirrored by a corresponding object in the compiled hierarchy. There is a lot of information about how to define OTcl scripts and run them in the NS, so we will only explain the basic ideas to set up a simple simulation environment.

- The first step is to initialize the simulator and open the output files which could be trace files (which contain the data from the simulation) or Network AniMator files (files used for visualization). They are called NAM files due to the name of the application which generates the visualization files in the NS is called NAM.

- We also need to define the finish procedure not only to terminate the program but also to close the output files. This finish procedure will be used at the end of the program and requires specifying the time when the termination should occur.

- The next step is to define the nodes where the protocol stack will run. Those nodes can be fixed or mobile. To define the nodes we need to set up the link between them (in case we are not in a wireless network), their position (important for the NAM files), movement (if it is a mobile node) and queues associated to them.

- Once we have set up the topology of our network we must define the protocols (which are called agents in the simulator) and applications that each node uses. When defining the protocol stack we do not only establish which protocol is used in each layer of the protocol stack but we also define the characteristics of each protocol layer (by giving values to different variables). In fact that is really important because it allows us to run the same simulation environment, also called Tcl script, with different characteristics.

- Finally we must schedule the events. As we said before the NS is a discrete event simulator which means that we have to set up when the events/processes begin and finish.

There are some differences depending if we are working in a wired or a wireless network. In our case we will focus in wireless networks. In Figure 2.1 an example of a simple wireless simulation environment, where we can see the characteristics explained above, is shown.

#define options

```
set val(chan)      Channel/wirelessChannel      ;#channel type
set val(prop)      Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)      Phy/WirelessPhy               ;# network interface type
set val(mac)       Mac/802_11                     ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)          LL                                ;# link layer type
set val(ant)        Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)     50                                ;# max packet in ifq
set val(nn)         2                                 ;# number of mobile nodes
set val(rp)         DSDV                            ;# routing protocol
set val(x)          500                               ;# X dimension of topography
set val(y)          400                               ;# Y dimension of topography
set val(stop)      150                               ;# time of simulation end
```

Here we define the lower layers of our protocol stack

#Initialize the simulator

```
set ns              [new Simulator]
set tracefd        [open simple.tr w]
set namtrace    [open simple2.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
```

#Define the finish procedure

```
proc finish {} {
  global ns tracefd namtrace
  $ns flush-trace
  #close the data files
  close $tracefd
  close $namtrace
  #execute nam file
  exec simple2.nam &
  exit 0
}
```

# configure the nodes

```
$ns node-config   -adhocRouting $val(rp) \
                          -llType $val(ll) \
                          -macType $val(mac) \
                          -ifqType $val(ifq) \
                          -ifqLen $val(ifqlen) \
                          -antType $val(ant) \
                          -propType $val(prop) \
                          -phyType $val(netif) \
                          -channelType $val(chan) \
                          -topoInstance $topo \
                          -agentTrace ON \
                          -routerTrace ON \
                          -macTrace ON \
                          -phyTrace ON \
                          -movementTrace ON
```

It is possible to set up which traces we want to obtain in the trace file

```
for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
}
```

We associate the node configuration with each node

```
# Provide the initial location of mobile nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 10.0
$node_(1) set Y_ 10.0
$node_(1) set Z_   0.0

# Define the movements of the nodes:
$ns at  5.0 “”$node_(0) setdest 200.0 200.0 2.0"
$ns at 10.0 “”$node_(1) setdest 300.0 300.0 1.0"
```

When the number of implemented nodes is large and the movement of the nodes is fast, we can define the initial location and the movements in a text file instead of writing all the information in the Tcl code.
This text file should be loaded in this point.

```
# Set the agent in each node:
set agent_(0) [new Agent/PBC]
set agent_(1) [new Agent/PBC]
$ns attach-agent $node_(0) $agent_(0)
$ns attach-agent $node_(1) $agent_(1)
```

Finally we should define the upper layers of our protocol. We must attach them to our nodes.

```
$agent_(0) set payloadSize 10000
$agent_(0) set modulationScheme 1
$agent_(0) periodicBroadcast ON

$agent_(1) set payloadSize 10000
$agent_(1) set modulationScheme 1
$agent_(1) set periodicBroadcastVariance 0.05
$agent_(1) set periodicBroadcastVInterval 0.001
$agent_(1) periodiBroadcast OFF
```

Here we define the variables and commands associated to the agent. When a variable is not defined the default value is used.

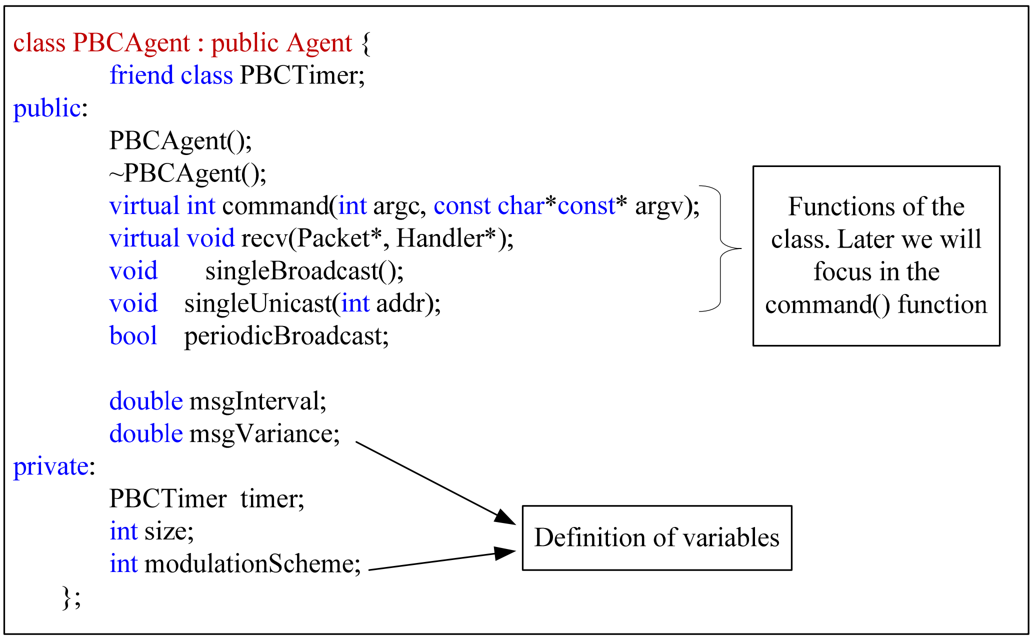
```
#define the initial node position for the nam
for {set i 0} {$i <$val(nn)} { incr i } {
$ns initial_node_pos $node_($i)  30
}
#Tell the nodes when the simulation ends
for {set i 0} {$i <$val(nn)} { incr i } {
$ns at $val(stop)  “$node_($i) reset”;
}
#And finally end the simulator
$ns at $val(stop) “finish”
$ns at $val(stop).01 “puts \”end simulation…\”; $ns halt”
$ns run
```

Figure 2.1: Example of a TCL script that simulates a wireless network.

We are interested in how to define some simulation environments and to obtain results and not only to debug our protocol implementation. But we must keep in mind that our main work is going to be related to the definition and implementation of a new protocol (which is not included in the actual version of the simulator) which means we will basically work in C++.

There is not a lot of information about how to define a new protocol in NS, but the basic ideas can be found in chapter VII of [31]. In this document it is explained how to set up a new protocol in NS by using an example. In our case we will show the main ideas by using an example, the PBC (Periodic BroadCast) protocol, a network protocol which is already included in the downloaded version of NS. We decided to use this protocol because our implementation will be slightly based on it. Instead of reproducing its content we are going to give some useful tips:

- In the header file we always have to declare our new class (called PBC Agent) as a subclass of the class "agent", including all the functions and variables we need:



```
class PBCAgent : public Agent {
        friend class PBCTimer;
public:
        PBCAgent();
        ~PBCAgent();
        virtual int command(int argc, const char*const* argv);
        virtual void recv(Packet*, Handler*);
        void     singleBroadcast();
        void   singleUnicast(int addr);
        bool   periodicBroadcast;

        double msgInterval;
        double msgVariance;
private:
        PBCTimer  timer;
        int size;
        int modulationScheme;
    };
```

Functions of the class. Later we will focus in the command() function

Definition of variables

- Sometimes when defining a new protocol we will also need to define a new type of header (and hence a new type of packet). In this case we have to declare, in the header file, the data structure of the new packet header:

```
struct hdr_pbc {
        double   send_time;
        static int offset_;
        inline static int& offset() { return offset_; }
        inline static hdr_pbc* access(const Packet* p) {
                return (hdr_pbc*) p->access(offset_);
        }
};
```

- We have to link the C++ code with the Tcl code, this is done declaring our class as an extension of the Tcl class:

```
static class PBCClass : public TclClass {
public:
        PBCClass() : TclClass("Agent/PBC") {}
        TclObject* create(int, const char*const*) {
                return (new PBCAgent());
        }
} class_pbc;
```

- In the case when we have defined a new header type, we must declare the new header as an extension of the Packet Header class:

```
static class PBCHeaderClass : public PacketHeaderClass {
public:
        PBCHeaderClass() : PacketHeaderClass("PacketHeader/PBC",
                                        sizeof(hdr_pbc)) {
                bind_offset(&hdr_pbc::offset_);
        }
} class_pbchdr;
```

- When introducing a new type of packet we will also need to modify two NS source files: the *packet.h* file and the *ns-packet.tcl* file. In the packet.h file (which in our downloaded version belongs to the ns-2.33/common/ file) we have to give an identifier and a name to our new header type:

```
typedef unsigned int packet_t;
static const packet_t PT_TCP = 0;
static const packet_t PT_UDP = 1;
…                                                  Here we give an identifier
static const packet_t PT_PBC = 45;  ──────────►       to our protocol
…
static packet_t     PT_NTYPE = 61; // This MUST be the LAST one
…
static void initName()
        {       …
                name_[PT_TCP]= "tcp";
                name_[PT_UDP]= "udp";
                …                                    Here we give a name to our
                name_[PT_PBC] = "PBC"; ──────────►            protocol
                …
}
```

- This step is slighly different as it is described in [31] due to our newer version of NS. In the *ns-packet.tcl* file we must add an entry for the new packets:

```
# Application-Layer Protocols:
        Message # a protocol to carry text messages
        Ping    # Ping
        PBC     # PBC
```

- After linking both classes we need to bind the variables defined in the Tcl code with the ones used in the C++ implementation, this step is done inside of the constructor of the class:

```
PBCAgent::PBCAgent() : Agent(PT_PBC), timer(this)
{
 bind("payloadSize", &size);
 bind("periodicBroadcastVariance", &msgVariance);
 bind("periodicBroadcastInterval", &msgInterval);
 bind("modulationScheme",&modulationScheme);
 periodicBroadcast = false;
 }
```

We need to include all those variables defined in the *ns-default.tcl* file (which is found in *ns-2.33/tcl/lib/* directory):

```
Agent/PBC set payloadSize 200
Agent/PBC set periodicBroadcastInterval 1
Agent/PBC set periodicBroadcastVariance 0.1
Agent/PBC set modulationScheme 0
```

Those default values will be used in case when we do not define the variables in the Tcl code.

- The last important thing we have to do, in order to define our protocol correctly, is to define the function *command(),* which is called when a Tcl command for the implemented class is executed. Usually those Tcl commands are used to start or stop the execution. If in a Tcl the used command is not found in the class command function, the same command is passed to the function of the base class.

```
int PBCAgent::command(int argc, const char*const* argv)
{       …
        if (argc == 3)
        {       …
                if(strcmp(argv[1],"PeriodicBroadcast") ==0)
                 {
                  if (strcmp(argv[2],"ON") == 0 )
                    {
                     periodicBroadcast = true;
                     timer.start();
                     return TCL_OK;
                    }
                  if(strcmp(argv[2],"OFF") ==0 )
                    {
                     periodicBroadcast = false;
                     timer.stop();
                     return TCL_OK;
                    }
                 }

return (Agent::command(argc, argv));
}
```

Here we show the two commands that are defined for PBC agent.

If the command hasn't been processed by PBCAgent()::command, call the command() function for the base class

- The last step is to add the file *pbc.o* in the makefile:
  *apps.pbc.o *

For work we did not only need a simulator, but also some documentation which defines the main, and sometimes also the specific ideas, necessary to develop our protocol theoretically. As it is mentioned in Chapter 1 there are two standards families involved in our diploma: IEEE 1609 and IEEE 802.11 standards.

The IEEE 1609 is a family of standards made for Wireless Access in Vehicular Environments (WAVE) and sponsored by the Intelligent Transportation Systems Committee of the IEEE Vehicular Technology Society. They are in charge of defining the architecture, communications model, management structure, security mechanisms and physical access for wireless communications in a vehicular environment [32]. This family of standards is formed by the following standards:

- IEEE 1609.1: Standard for Wireless Access in Vehicular Environments (WAVE)-Resource manager. Defines the services and interfaces of the WAVE resource management application.

- IEEE 1609.2: Standard for Wireless Access in Vehicular Environments (WAVE)-Security services for applications and management messages. This standard is in charge of defining all the security processes and mechanisms to avoid spoofing, eavesdropping and so on.

- IEEE 1609.3: Standard for Wireless Access in Vehicular Environments (WAVE)-Networking services. It defines the transport and network layer services of the data plane of the WAVE protocol stack. As it is shown in Figure 1.1 it also defines the Management Information Base (MIB) which belongs to the management plane of the protocol stack.

- IEEE 1609.4: Standard for Wireless Access in Vehicular Environments (WAVE)-Multi-channel operations. Standard in charge of adapting the MAC layer (defined in IEEE 802.11 standard) to support WAVE communications. It is also described how to handle different communication channels (control and service channels); reason why we will deeply use this standard.

A fifth standard, IEEE 1609.0, is underway as an architecture document that will give an overview of WAVE systems and their components and operation, as well as a context to better understand the content of other WAVE standards and IEEE 802.11 (WAVE mode).

The other family of standards is well known in wireless communication world: the IEEE 802.11 standards. The first Wireless Local Area Network (WLAN) standard, IEEE 802.11 was adopted in 1997. This standard defined the MAC and PHY layers for a LAN with wireless connectivity. It addresses local area networking where the connected devices communicate over the air to other devices that are within close proximity to each other. Since 1997, the IEEE 802.11 standard working group has been extended with numerous task groups; designated by different letters and orientated to different areas (for example IEEE 802.11a defines WLAN operations in the 5 GHz band, with data rates of up to 54 Mbps). From all of these task groups we use the recent defined IEEE 802.11p which is a draft standard that specifies the extensions to IEEE 802.11 for Wireless Local Area Networks providing wireless communications while in a vehicular environment [33].

3 Protocol Explanations

As we explained before, we want to see if TDMA is better than FDMA when multiplexing the Control and the Service channel. Before being able to compare both multiplexing technologies we need to define theoretically and implement (by using the Network Simulator) the TDMA protocol. In this chapter we are going to explain the main ideas used later to develop the TDMA protocol in the NS.

Our protocol is going to be a provider-client protocol, this means the protocol is centralised, the other possibility is to define a distributed or an ad hoc protocol as it is done in [7] and [8]. Centralized means the provider will be the only one who handles the information given in both channels. This does not mean that the communication will be only unidirectional (from the provider to the client), but we will see that sometimes a bidirectional communication is necessary. The provider in our case is the RSU although it could be also any OBU. Therefore all clients are going to be OBUs. From now on we will always use the term provider and client to refer the RSU and the OBU respectively.

We decided to implement frames of 100 ms, these frames will have two main time slots one for the control and another for the service channel both of the same duration (50 ms) [1] and [2]. We chose the value of 50 ms because this is currently used in FDMA implementation.

We will spend all the time in the control channel time slot to send non-safety information. This information is basically data about the services, which are going to be offered in the service channel time slot such as the subtime slot, where the service is going to be used, the identifier of the service or the type of service offered (type in our case means if the service is going to be broadcast or unicast).

We should not forget that the control channel is not only used to send general or non-safety information that the clients need to receive properly the service they are interested in but also to send critical data. This critical data is a high priority flow, used to avoid dangerous situations. An example of this data could be a message alerting that the car which is in front of you has just stopped or that the traffic light is going to turn red in a few seconds.

We decided that we will not implement those kind of messages, because the main aim of our implementation is to compare with a simple FDMA. Introducing high priority frames makes the protocol very complex for several reasons:

- High priority frames means that they should be sent even if we are in the service channel time slot. That requires the TDMA frame to be flexible: the control channel time slot duration will be constant if there are no emergency frames (then only information about the available services must be given) but in the case that there is an emergency the control channel time slot should be as long as the emergency is. Obviously developing a flexible TDMA protocol is quite complex and it is out of the scope of our diploma.

- Also with the actual implementation this is not possible: there is only one provider which has to handle both types of frames (for the control and service channel) for all the clients which belong to its coverage area. This means if a client needs to receive emergency frames the other clients will have to wait till this process is finished. This is not really useful if we want to implement a dynamic protocol.

- If we are interested in offering emergency frames the easier solution could be to define the time slot duration constant and, in case an emergency process appears when we are in the service channel time slot, to postpone sending the emergency frames until we reach the following control channel time slot. This is a good option in case the service channel time slot duration is not too long. Anyway, having a fixed duration of the control and service channel is the option that is used in FDMA based WAVE devices nowadays.

Because we will use the control channel time slot to send only non-safety information and this information is supposed to reach all the clients which belong to the area of coverage of the provider, we decided that those frames will be broadcast frames. Also in this time slot the communication will be unidirectional which means the clients will not acknowledge the frames received. We do not consider important to acknowledge the frames because the information given is just non-safety information. Another reason is that if we want all the clients to acknowledge the frames received during the control channel time slot the provider will probably have to wait for a long time making the protocol to be slowly.

If we focus in the service channel time slot the protocol gets complex. During this time slot the provider sends the data related to the different services available. Usually the provider will have more than one service to offer. To be able to send information about more than one service we must divide the service time slot into different "subtime" slots, each of them for a different service. It is important to pay attention that we divide the service channel time slot into as many subslots as services the provider wants to offer, not as many clients are in the area [7] and [16], fact that it is shown in Figure 3.1. We considered that this option is better, because of two reasons:

- There would probably be few services to offer than clients, so we have bigger subslots and hence more time for each service.

- Some of the services offered will be broadcast services, in this case our option is more efficient because in the case that we assign one subslot time to each client if more than one client desires the same broadcast service, we will have to send the same information as many times as clients want it. This means that although we are sending broadcast frames, only one client will receive it at any time, so we would lose important time sending broadcast information which has already be sent.

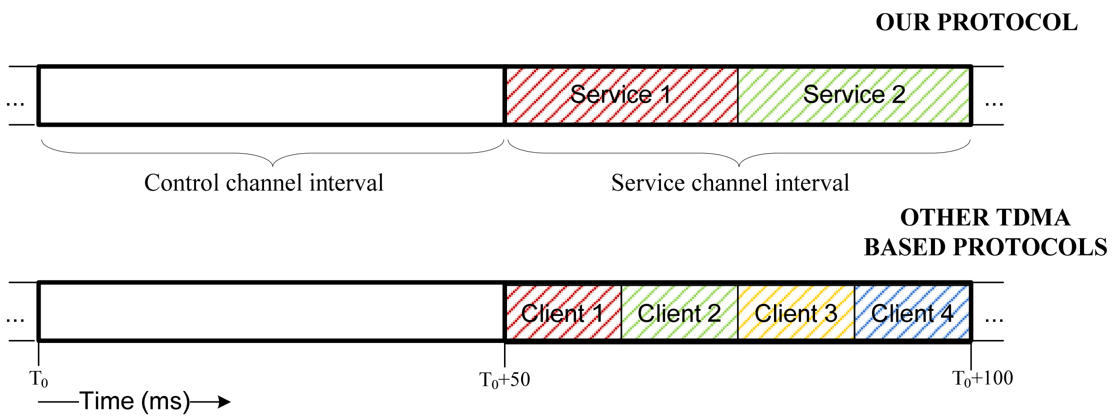


Figure 3.1: Comparison of different ways of using the service channel interval.

Assigning a sub slot to each service also has some disadvantages; for example in case a unicast service is offered we will have to deal with the problem of more than one client trying to receive the information, in this case CSMA/CA should be used within the TDMA.

As mentioned above there are two types of services: unicast and broadcast. An example of broadcast service could be a forecast service where the client gets information about the weather or traffic information.. A unicast service could be to pay the toll in a motorway in a wireless way.

In our case, when a broadcast service is offered, the communication between provider and client is unidirectional, which means that the client will not acknowledge the frames received from the provider. In this situation we decided to have no bidirectional communication just for the same reason than in the control channel time slot. Therefore when the subtime slot for the unicast service begins, the provider will send immediately data broadcast frames. The clients will detect those frames and receive or discard them depending if they are interested or not in the service.

When a unicast service is offered, the communication between provider and client needs to be bidirectional basically for two reasons:

- The provider needs to know the MAC address of the client before sending the unicast frames. The question is when and how the provider gets the information of the addresses of all the clients which belong to its coverage area. One possibility is that it gets this information during the synchronization process that provider and client carry on when the last one enters in the coverage area of the provider. For certain reasons (see details in Chapter 5) this process is not implemented in our protocol.
  Another possibility is that the provider gets the information of the address of the client just at the beginning of the subtime slot where the unicast service is offered. So we need the client to communicate (through a message) its address to the provider, that reason makes the communication bidirectional.

- The second reason is even more important than the first one. If we assume that the provider knows all the addresses of all the clients (probably the provider has a table with this information) maybe we can think we do not need a bidirectional communication between provider and client but we forget one important fact: the provider is not able to guess if a client is interested or not in the unicast service. So we need the client to "tell" the provider if it is interested in the service. For this reason we need a bidirectional communication.

Once we have seen why we need to set up a bidirectional communication, we need to explain more in detail how this communication will work. The basic idea consists in two steps: at the beginning of the subslot the client communicates (through a small frame that we will call request frame) to the provider that it is interested in the service and gives its address. Once the provider has this information and the time to receive request frames has expired it begins to send the data to the client until the subslot finishes.

Although the process seems to be simple in fact there are a lot of points that must be considered. One fact is, what happens if the frame sent by the client is lost. Although the probability of losing a frame is not high (assuming the channel does not introduce big loses) the situation should happen sometimes. In our simulator we assume the frames will never get lost, because this is one of the main characteristics of TDMA technology, this means the client only will need to send one frame at the beginning of the subslot.

Another fact is what happens if more than one client is interested in the same unicast service. As we said before we associate the subslots (in the service channel) to the services not to the clients, which means if two or more clients are interested in the same service they will share the same time slot. If the service is broadcast there is no problem, but if it is unicast we need to handle the access of different clients to the medium, in order to avoid possible collisions.

The first idea, which comes up in our mind is to use CSMA/CA as in IEEE 802.11a protocol. Basically a CSMA protocol consists on the following process: the station which wants to transmit data needs at first to sense the medium, if it is free then it sends the data but if it is busy then the transmission is postponed to a later time (a backoff algorithm is used to retransmit the data). When CSMA works together with CA (Collision Avoidance) the protocol gets more complex: if the medium is busy the process is the same but when the medium is free the station will not transmit immediately. In this case the station will be able to transmit the data if the medium is free during a specific time (which is called DIFS: Distributed Inter Frame Space). The receiving station will then acknowledge the data received. If the transmitter does not receive any acknowledgement it will retransmit the data until it receives an acknowledgement or will stop retransmitting it after a certain number of retransmissions [34].

In CSMA there is also another mechanism used to reduce the probability of two stations colliding when they cannot listen each other, this mechanism is called Virtual Carrier Sense. The Virtual Carrier Sense requires the transmitter to use short control packets called RTS (Request To Send) and the receiver to use response control packets called CTS (Clear To Send). Figure 3.2 shows how the mechanism works.

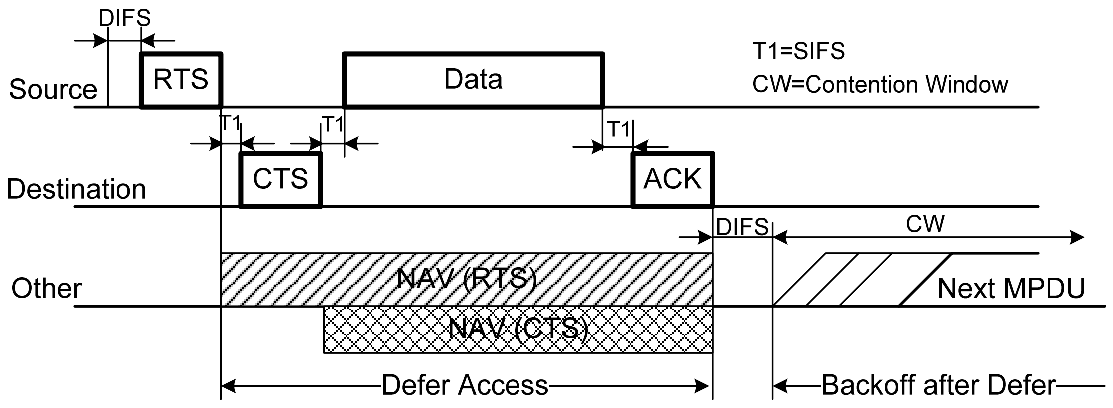


Figure 3.2: Process followed in a CSMA/CA with Virtual Carrier Sense process between two wireless nodes. Figure based in [34] and [35].

The IEEE 802.11 protocol shows how CSMA works in detail (see chapter 9 of [36]). We cannot apply exactly the ideas explained before due to the following reason: in CSMA/CA the transmitter of the data is the one that needs to sense the medium before sending its information, in our case the receiver is the one which needs to do that. If we think about this process shown in Figure 3.2 and try to adapt it to our protocol: who must be the one which sends the RTS packet: the transmitter/provider or the receiver/client? This question is not so easy to answer, because we must avoid collision between frames sent by the clients not between the ones sent by transmitters.

We decided to simplify the process: if more than one client wants to receive data from the same unicast service and hence must send a small packet to the transmitter to communicate its address and interest in the service, the client will sense the medium first and if empty, will send the packet with its address. If the medium is busy the client will wait until it is empty and send the packet then.

If more than one client is interested in the same service and the service cannot be consumed by different clients at the same time, because it is unicast, we must define the total size of the data of each service. We should realise that probably the provider will need more than one packet to send all the information about the service. We have to define (if we want to avoid having to implement a fragmentation mechanism) the number of packets necessary to consume each unicast service. Then, when one client A has just finished consuming the service the client B is able to begin consuming it.

When defining the size of each service data (which will probably be bigger than the size of the packets generated by the application layer as we said before) we establish a kind of continuity between frames. That means that if a client is not able to consume the service in one frame it will continue receiving the packets in the next service timeslot.

As we said before the clients will only send a request frame to the provider at the beginning of the subtime slot where the service is offered. In case a client needs more than one frame to consume an unicast service, it will not send a request frame in the following sub slots because the provider already knows the client has not finished consuming the service. In case the client does not receive any data packet as a response of the request frame it will assume there is another client already consuming it (the probability of losing the frame is depreciable). In this situation this client will send again a request frame in the next service channel subtime slot in order to indicate the provider that it is still interested in this service. A client could finish consuming a unicast service before the subtime slot expires. That means the provider has some time left. Instead of wasting this time the provider would begin sending data unicast packets to another client (in case the provider has received request frames before).

Those ideas can be appreciated in Figure 3.3, where we can see in the first unicast interval both clients send a request frame to the provider (RSU). There is no collision between both request frames. The provider will attend at first the OBU 1 because its request frame arrived first. As it is said in the square at the right top of the figure the unicast service offered by the provider requires five frames to be consumed. This is the reason why the OBU 1 will require two service channel slots to consume the service. If we pay attention to the second SCH interval, we can see how once the provider sends the last two frames to the OBU 1 there is some time left. Instead of wasting this time doing nothing, the provider begins sending frames to the OBU 2, because it already knows OBU 2 is interested in the service. Another important fact is to point out that the OBU 1 does not send any request frame in the second SCH interval, because it already begun consuming the service in the first SCH interval.

In Figure 3.4 an example of how two services (one unicast and the other broadcast) are multiplexed in the service channel time slot, is shown. In this case the service channel time slot is divided into two sub-time slots, the first one will be destined to offer the unicast service. We can appreciate how in case of a broadcast service the client does not ask, through a request frame, the provider for the service (main reason why we call those services broadcast ones)

We explained before the problem of possible collisions if more than one client is interested in the same unicast service. But there are also other situations where collisions could happen. Basically these collisions are produced when we change from control channel timeslot to service channel timeslot (and vice versa) or when we change from one service subslot to another service subslot. We will give an example to understand it better:

If we assume that there is only one service offered by the provider and it is unicast. There is only one client interested in the service (so we will not have problems of collisions between clients). If the last packet from the control channel is sent just few microseconds before the timeslot changes and the client sends the request packet really early in the service channel slot then a collision will occur. Why? Because the client decides to send the request packet before receiving the last packet of the control channel (because it does not know a packet is missing) or saying it the other way around, the provider receives the request packet before finishing sending the last packet of the control channel.

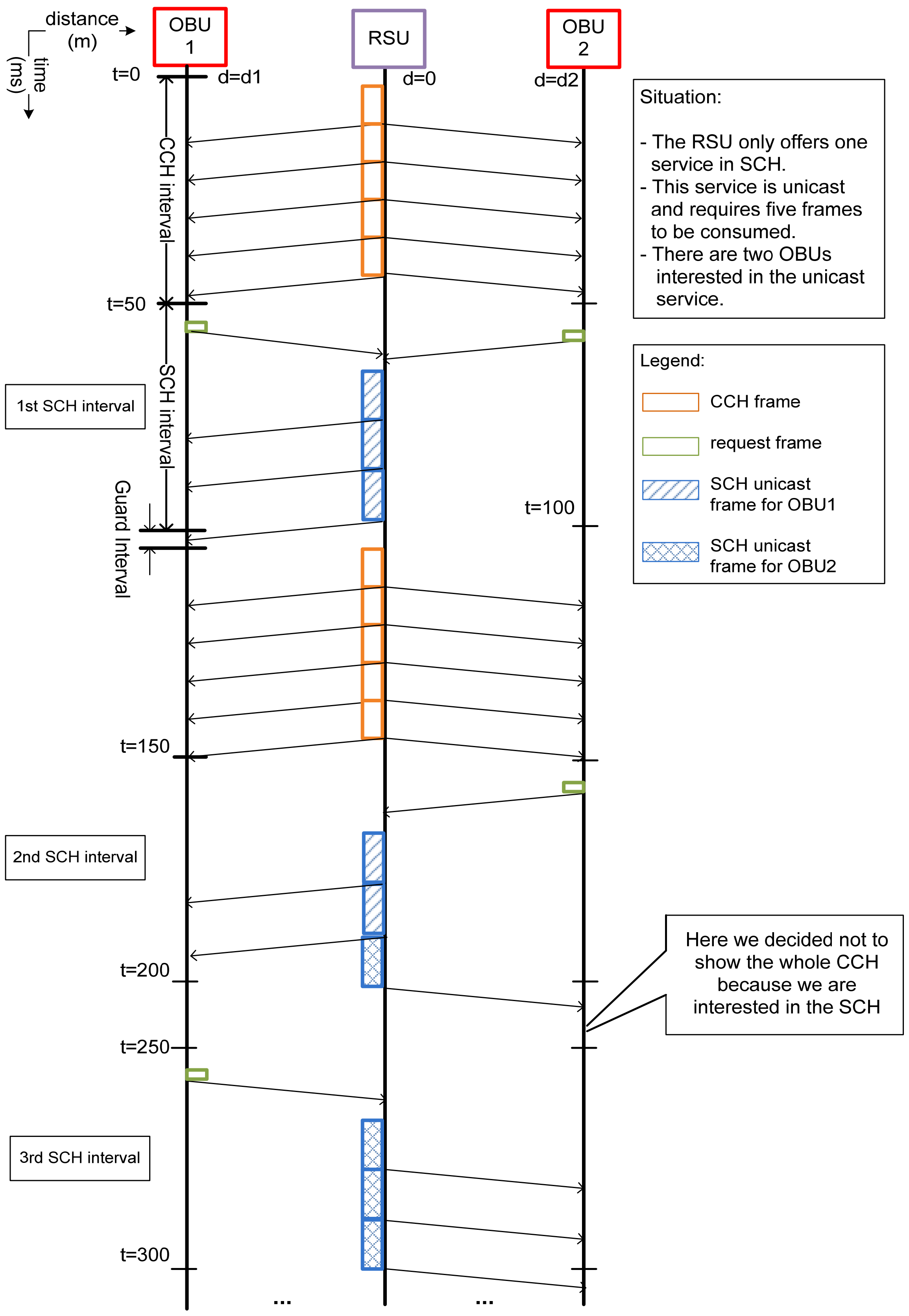

distance (m)
time (ms)
OBU 1
RSU
OBU 2
t=0
d=d1
d=0
d=d2
CCH interval
t=50
SCH interval
1st SCH interval
Guard Interval
t=100
t=150
2nd SCH interval
t=200
t=250
3rd SCH interval
t=300
...
...
Situation:
- The RSU only offers one service in SCH.
- This service is unicast and requires five frames to be consumed.
- There are two OBUs interested in the unicast service.
Legend:
CCH frame
request frame
SCH unicast frame for OBU1
SCH unicast frame for OBU2
Here we decided not to show the whole CCH because we are interested in the SCH

Figure 3.3: Simple example of how the same unicast information is multiplexed between two clients (OBUs).

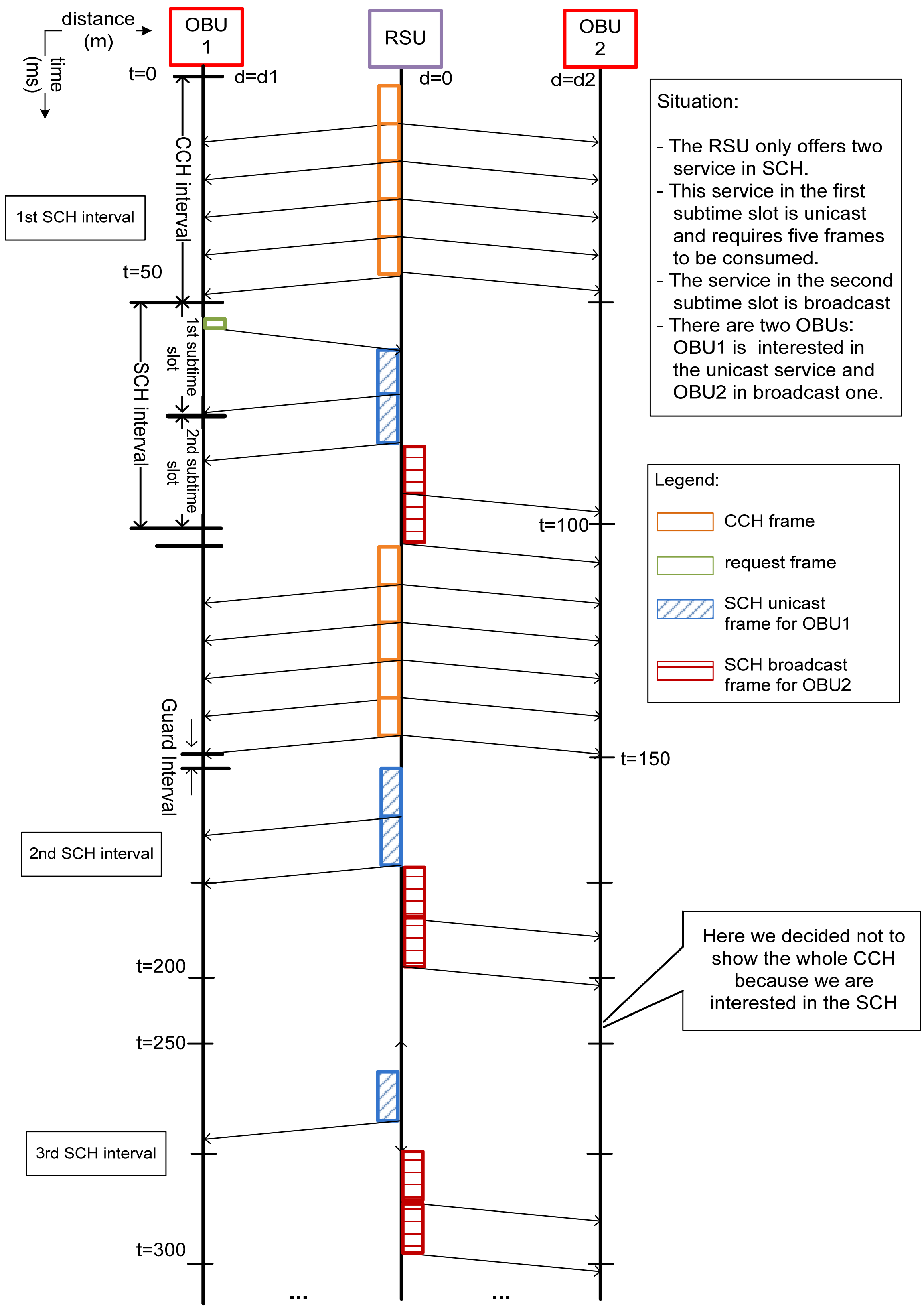

distance (m)
time (ms)
OBU 1
RSU
OBU 2
t=0
d=d1
d=0
d=d2
CCH interval
1st SCH interval
t=50
SCH interval
1st subtime slot
2nd subtime slot
t=100
Guard Interval
t=150
2nd SCH interval
t=200
t=250
3rd SCH interval
t=300
...
...
Situation:
- The RSU only offers two service in SCH.
- This service in the first subtime slot is unicast and requires five frames to be consumed.
- The service in the second subtime slot is broadcast
- There are two OBUs: OBU1 is interested in the unicast service and OBU2 in broadcast one.
Legend:
CCH frame
request frame
SCH unicast frame for OBU1
SCH broadcast frame for OBU2
Here we decided not to show the whole CCH because we are interested in the SCH

Figure 3.4: Simple example of how a broadcast service and a unicast service are multiplexed between two clients (OBUs).

The reason why these collisions occur is when sending a packet we only take into account the time the packet is began to send (if this time is smaller than the limit which indicates the ending of the time slot of subtime slot then the packet is sent otherwise it is not), but not the time the packet needs to be transmitted. This fact makes collisions occur, if the packet is transmitted really close to the ending of the slot time. To avoid these situations we decided to leave an empty time at the end of any time slot (and subtime slot). From now on we can talk about the theoretical slot time (which is for the control and service channel 50 ms) and the real time which will be the theoretical time minus this "guard time".

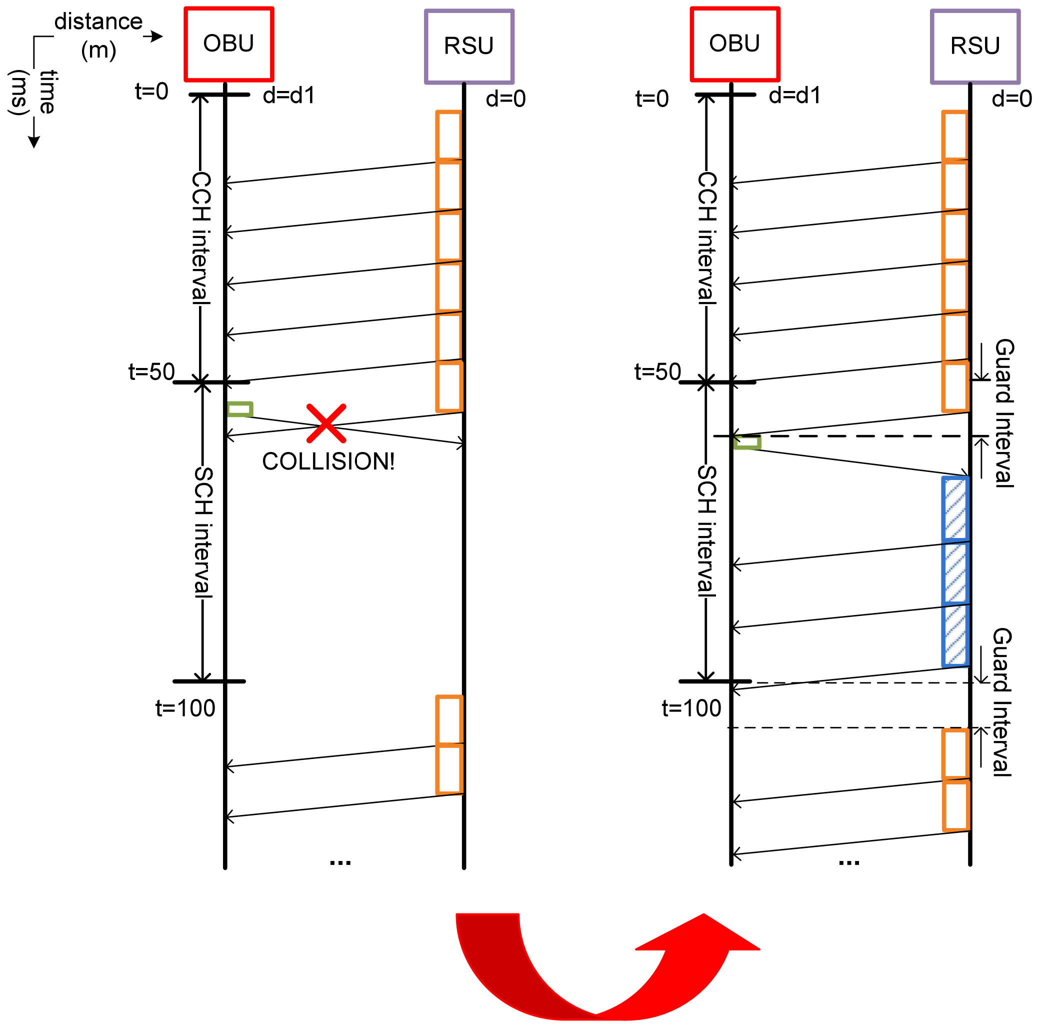


Figure 3 5: Example which shows why it is necessary to introduce guard intervals.

Once we have set up the main characteristics of our protocol we must define the specific details by using the protocols IEEE 1609.4 and IEEE 802.11p.

From the IEEE 802.11 protocol we got information about the definition of the MAC frames and about characteristics of the physical layer. The IEEE 802.11 standard establishes the general format of a MAC frame which consists of three fields: the MAC header, the body and the Frame Check Sequence (FCS). The header (as we can see in Figure 3.6) is divided into several fields. The existence of all of those fields depends on the type of the MAC frame.

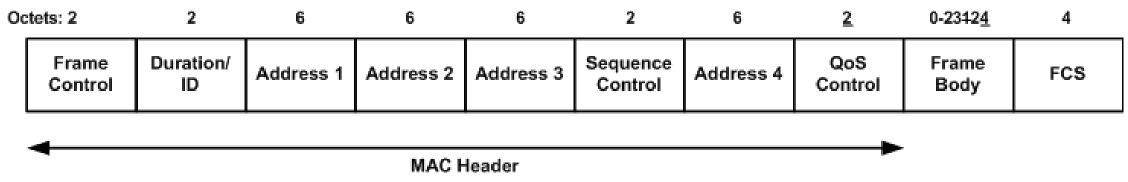


Figure 3.6: Fields of a MAC frame defined in [36]. We can appreciate the three main fields: the MAC header, the frame body and the FCS.

The smallest MAC frame is defined by the Frame control, Duration/ID, Address 1 (which belong to the header) and the FCS (page 60 on [36]). If we pay attention to the frame control field we will see that it has the following fields:

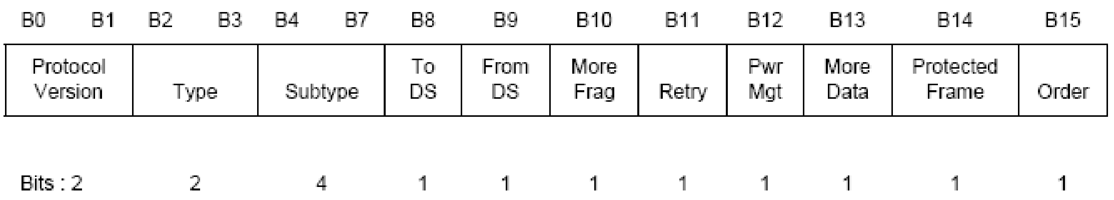


Figure 3.7: Fields of the frame control field included in the MAC header [36].

The field type allows the definition of three types of MAC frames: management, data and control frames. In [36] (table 7-1) different subtypes of each type of frame are defined. We need to know which type of MAC frame we need to use for sending the information in each channel: CCH and SCH. As mentioned in Chapter 1 and explained at the beginning of this chapter, the service channel will be used to send application data and the control channel to announce WAVE services. For this reason we decided that in SCH we will transmit data frames (subtype data) and in CCH we will transmit management frames (subtype beacon). Further details of the frame formats are defined in chapter 7 of [36].

From the standard IEEE 1609.4 we got the idea about how to define the mechanisms or services used to support MAC Service Data Unit (MSDU) delivery and to manage channel coordination. Those services constitute an extension of the functions introduced in protocol IEEE 802.11 and are basically necessary to enable multi-channel coordination (page 8 in [3]). Those services are:

- Channel routing: This service controls the routing of data packets from the Logic Link Control (LLC) to the MAC. The process will be different depending on the data we want to route: WAVE Short Message Protocol (WSMP) data or IP data.

- User priority: Once an MSDU arrives at the MAC layer and the channel routing process has been done, the User Priority (UP) is used to handle MSDUs with different priority. The Enhanced Distributed Channel Access (EDCA) functionality is used. This user priority is necessary to support a variety of safety and non-safety applications.

  We should point out that the goal of having the EDCA functionality is to handle the access to the medium when CSMA/CA is used, because each user receives a different priority in the access to the medium. In our case this module should be adapted to the TDMA scheme, which means there is no priority between users, but a priority between different data flow corresponding too different applications in the provider side.

- Channel coordination: This service is implemented to support data exchanges between one or more devices, which are not able of simultaneously monitor the CCH and exchanging data on SCHs. This service requires a synchronization procedure defined in [36] (page 13).

- MSDU data transfer: this service is in charge of sending the data which belong to the CCH or to the SCH.

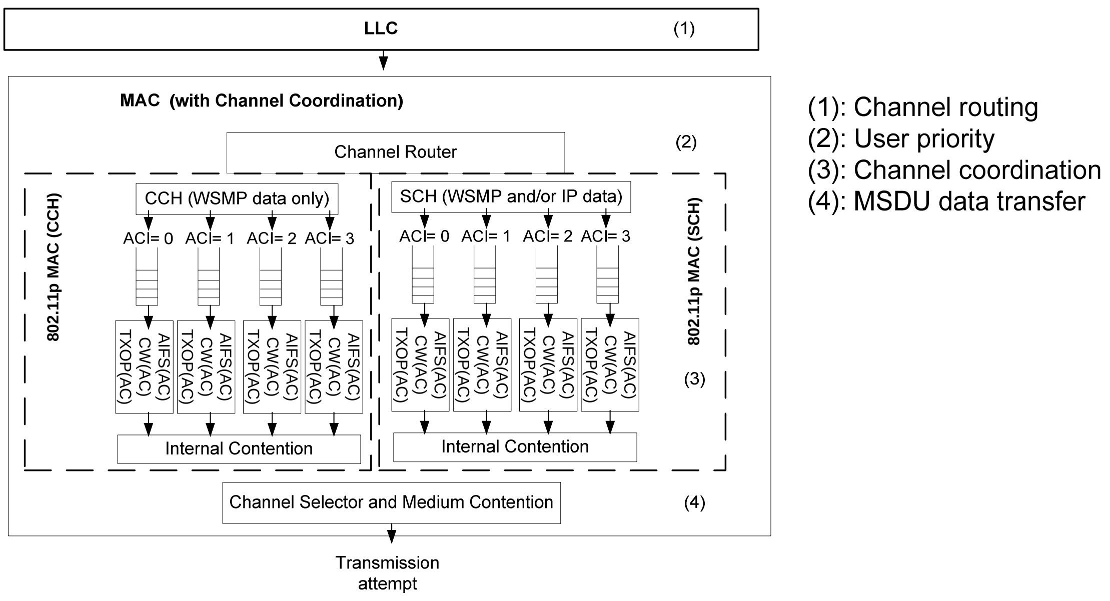


Figure 3.8: Architecture of MAC layer in WAVE devices. Image from [3].

As we can see in Figure 1.1 the MAC and PHY layers include management entities (called MAC Layer Management Entity, MLME, and Physical Layer Management Entity, PLME, respectively). These management entities provide the layer management service interfaces through which layer management functions may be invoked (page 15 in [3]). The WAVE Management Entity (WME) is a layer-independent entity which would typically perform such functions on behalf of general system management entities and would implement standard management protocols.

If we want to implement our protocol correctly, we should define both protocol planes and all the processes which communicate both planes, the only problem is nowadays the management plane is not implemented in the Network Simulator, which means there is a lot of work to do. Finally we opted to do only changes related to the data plane. Although the scope of this diploma thesis is to define a MAC protocol we will probably have to deal with the physical layer. Basically we need to set up correctly the parameters which define the physical layer in C2X communications. Those parameters are the modulation, the bandwidth (or data rate), the definition of the transmission time and the sensitivity of the receiver.

We decided to base our analysis of the protocol assuming we are working with channels of 10 MHz although 20 MHz and 5 MHz channels are also defined in the

IEEE 802.11 standards. All the values are obtained from [36] (chapter 17) and summarized in the following Table 3.1:

Modulation	Coding Rate (R)	Coded bits per subcarrier (NBPSC)	Coded bits per OFDM symbol (NCBPS)	Data bits per OFDM symbol (NDBPS)	Data Rate (Mb/s)	Minimum sensitivity (dBm)
BPSK	1/2	1	48	24	3	-85
QPSK	1/2	2	96	48	6	-82
16-QAM	1/2	4	192	96	12	-77
64-QAM	3/4	6	288	216	27	-68

Table 3.1: Modulation- dependent parameters for 10MHz channel spacing. Table based on Table 17-3 and 17-13 of IEEE 802.11 standards.

In [36] (table 17-4) the timing-related parameters are defined, in the Table 3.2 we show the values given for 10 MHz channel spacing:

Paremeter	Value for 10 MHz channel spacing
N_{sd}: Number of data subcarriers	48
N_{sp}: Number of pilot subcarriers	4
N_{st}: Number of subcarriers, total	52 (N_{sd}+N_{sp})
Δf: Subcarrier frequency spacing	0.15625 MHz (=10 MHz/64)
T_{fft}: Inverse Fast Fourier(IFFT)/Fast Fourier Transform (FFT) period	6.4 μs (1/Δf)
$T_{preamble}$: PLCP preamble duration	32 μs (T_{short}+T_{long})
T_{signal}: Duration of the signal BPSK-OFDM symbol	8.0 μs (T_{gi}+T_{fft})
T_{gi}: GI duration	1.6 μs (T_{fft}/4)
T_{gi2}: Training symbol GI duration	3.2 μs (T_{fft}/2)
T_{sym}: Symbol interval	8 μs (T_{fft}+T_{gi})
T_{short}: Short Training symbol sequence duration	16 μs (10xT_{fft}/4)
T_{long}: Long Training symbol sequence duration	16 μs (T_{gi}+2xT_{fft})

Table 3.2: Time-related parameters for 10MHz channel spacing. Table based on Table 17-4 of IEEE 802.11 standards.

From this table we are able to define the transmission time which is really important when using TDMA technology:

$$TXTIME = T_{PREAMBLE} + T_{SIGNAL} + T_{SYM} \times Ceiling((16 + 8 \times LENGTH + 6) / N_{DBPS})$$

We are interested in having a exact definition of the transmission time because we will use it to define the guard time and to set up the maximum number of sub slots (and hence of services) in the service channel time slot.

To define the guard time, which is necessary to avoid collisions, we must know the time necessary to transmit a frame. In our implementation we set up frames of 1000 bytes of payload for both control and service channel. By using the above formula we will need the following transmission time:

$$t_{tx}\Big|_{data} = 32\mu s + 8\mu s + 8\mu s \times Ceiling((16 + 8 \times 1000 + 6) / 24) = 2.712ms$$

As mentioned before the collisions are produced when a frame is sent just few microseconds before the time slot expires. In our case for a payload of 1000 bytes we have a transmission time of 2.712 ms so we will avoid those collisions if we set up a guard time of 3 ms just at the end of each time slot or subtime slot.

We explained that in the case when a unicast service is offered it is necessary that the client sends a request frame to the provider indicating its address and its interest in the service. We will need also to define a time, at the beginning of the sub slot of the unicast service, when the clients are able to send the request frame without collisions. For a request frame of 50 bytes the transmission time is:

$$t_{tx}\Big|_{request} = 32\mu s + 8\mu s + 8\mu s \times Ceiling\left(\left(16 + 8 \times 50 + 6\right) / 24\right) = 0.176ms$$

Keeping in mind that probably more than one client sends a request frame, a reasonable option is to set up a “guard time” of 3 ms allowing a maximum of seventeen clients to send a request frame.

To define the maximum number of sub slots in the service channel interval we must analyse the time necessary to consume a service in the worst situation: when a unicast service is offered. If we consider the situation where a client wants to receive information about an unicast service of 5000 bytes and the payload of a data frame in 1000 bytes and of a request frame is 50 bytes, the time necessary to consume the service will be:

$$totalTX = 0.176 + 5 \times 2.712 = 13.736ms$$

We need about 14 ms to consume the service depreciating the propagation time (which will be about 2µsec for each frame considering a path distance of 500 m). If we have four subtime slots in the service channel interval we will have 50 ms / 4 slots = 12.5 ms for each service. But we must keep in mind the guard interval and the time destined to receive request frames, so in fact we have 12.5ms - 3ms - 3ms = 6.5 ms. In 6.5 ms the provider will be able to send two data frames to the client, and therefore will require three frames to attend the client, which is really a lot of time (about 2 x 100 ms + 50ms + 12.5 ms = 262.5 ms approximately). If there are only three subtime slots instead of four the time necessary to consume the service is reduced considerably: 50 ms / 3 slots ≈ 17 ms → 11 ms to consume the service, four data frames sent in each sub slot, 166 ms estimate to consume the whole data. So we decided the maximum number of sub slots in the service channel interval will be three.

4 Protocol Implementation

In the former chapter we described theoretically the main characteristics of the protocol we wanted to develop. In this chapter we are going to explain how we implemented these ideas by modifying the C++ code in the Network Simulator. Although the protocol we want to develop is basically a MAC protocol, we must keep in mind that not only changes in the MAC layer will be done. We will also have to deal with the physical and application layer.

From the point of view of the provider the MAC layer is the one which is in charge to handle the different types of packets (from control and service channel). In fact, the MAC layer in the provider side is the one which carries on the TDMA multiplex between both channels (and between different services inside the service channel). The application, in this case, will generate the packets which are going to be sent in each channel.

From the point of view of the client both layers, MAC and application, are simpler. The MAC layer is basically in charge of sending to the application the packets the client wants to receive (packets which belong to the control channel or from a service desired) or discarding the packets not requested by the client (packets from a broadcast or unicast service the client is not interested in). The application layer will be the one which generates the request packets the clients send to the provider when they are interested in a unicast service.

Although in Chapter 3 we described in detail the characteristics of the MAC protocol developed we must realise the idea described there corresponds to the last version of our protocol, to reach this implementation we programmed and tested before simple versions, the main changes and improvements done between versions are related to the definition of the different services offered in the service channel timeslot:

- The first version only considered a unidirectional communication between provider and client. The reason was that we only defined broadcast services in the service channel.

- A second version consisted in defining unicast services and hence introducing a bidirectional communication between provider and client. This new version was more complex than the previous one so we decided to divide the application we had until now (called pbc3) into two sides: the application in the provider side (pbc3) and the application in the client side (pbc3Sink). This idea of defining two "sides" of an application or protocol layer (to simplify its implementation) is already used in other applications or protocol layers included in the simulator as TCP.

- The third and last version consisted in implementing the algorithm which handles the access of more than one client to the same unicast service. This was not considered in version two.

When programming our MAC protocol several problems arised, which where not only on our side, but also limitations or restrictions from the Network Simulator. Although in Chapter 5 we describe and explain all of them, we need to mention one limitation in advance, because it deeply influenced the implementation of our protocol. The limitation consists on the fact of having more than one parallel flow of data in the same node (the node can be client or provider). In our case we decided to have only one provider, this provider will produce the data of both control and service channel. This means the provider will have more than one application running in parallel and more than one queue in the MAC layer (see Figure 3.7), each one associated to a different data flow. That is the reason why we are interested in parallelizing.

The problem is, as far as we know, setting up parallel flows in the same node is not simple to do in the simulator. The most common solution consists on using as many nodes as parallel flows, an idea used when a protocol stack is defined by two or more planes (like data and management plane shown in Figure 1.1). This is deeply explained in [37], or when we desire to have a multi-interface node, in [38].

The idea is: if we are not able to have more than one application running in parallel in the same node, what is the possible solution? The solution we took, there is another one explained in Chapter 5, is to have only one application running in the provider, which generates different types of packets depending on the execution time. This approximation also solves the problem of having more than one queue (in parallel) in the MAC layer; we will not need different queues to "store" different packets due to those packets arrive to the MAC layer already in the order they must be sent. This solution simplifies the definition of the MAC layer but makes the definition of the application layer to be more complex.

Although the solution we took could seem rudimentary the fact is the difference between the theoretical solution and the real one is not so big if we think about what we want to test once the protocol is implemented (see Chapter 6).

After mentioning and explaining this problem we can specify how we did our protocol. We will begin explaining how the application is defined in both, client and provider. Both "sides" of the application have two main functions: one which is in charge of creating and sending the packets to the lower layers and another, which is in charge of receiving the packets from the lower layer.

The application in the provider side is called pbc3 and has two main functions: one for sending and the other one for receiving frames. When sending frames we basically need to create a packet (by defining its header) and send it. The provider will send different types of packets depending on the execution time. Basically we will have two types of packets: management packets in the control channel timeslot and data packets in the service channel timeslot. Those packets will have different headers. In case of the data packets the fields of the header are the following:

type	Service_ id	Time_slot	seqNum	lastPacket	node_id	Send_time	payload

Figure 4.1 : Fields of the application header for data frames.

- Service_id: Field used by the provider to indicate the service whose payload is included in the frame.
- Time_slot: Field which shows the subtime slot when the service is offered.
- seqNum: Sequence number of the packet sent. Nowadays is only used in data packets which belong to unicast services, it is used by the provider when more than one client wants to receive the same private information.
- lastPacket: This field is related to seqNum. It is used to indicate that the packet sent is the last one.

In case of management frames the header is defined by the following fields:

type	Services_ Information	Num_ services	node_id	Send_time	payload

Figure 4. 2: Fields of the application header for management frames.

- services_information: It is only used in management frames. It is a vector which contains the basic information about the services offered by the provider. This basic information is defined by three fields: the first field is the identifier of the service, the second field is the subtimeslot identifier and the third field is the type of the service (as we said already before the type of the service means if the service is broadcast or if it is unicast). These three fields must be defined for each service available in the provider.

Service 1 identifier	Time slot where service 1 is offered	Type of service 1	Service 2 identifier	Time slot where service 2 is offered	Type of service 2

Figure 4.3: Example of the services_information buffer when two services offered.

Although in our implementation the identifier of the service and the identifier of the timeslot is the same (which means the service whose identifier is the number one will be offered in the subtime slot number one) we decided to define two variables because they would have different values in future versions of the protocol.

- Num_services: Value used to indicate the total number of services which are going to be offered by the provider during the service channel timeslot.

As we could realise there are three fields in the header which are common in data and management packets, those fields are:

- type: To indicate the type of the frame. There are two options: management frames used in control channel time slot and data frames used in service channel timeslot. This field of the header is used in the MAC layer to insert different MAC headers (as it is described in [36]).
- node_id: To indentify the node who sends the frame. The identifier of the provider will be always zero.
- Send_time: this value is used internally by the simulator; it is related to the temporization of the events.

Once we know the fields which define the application header of the frames the provider sends, we can explain the two main functions which define the application. One function is the *sendFrame()* and it works as it is shown in the Figure 4.4.

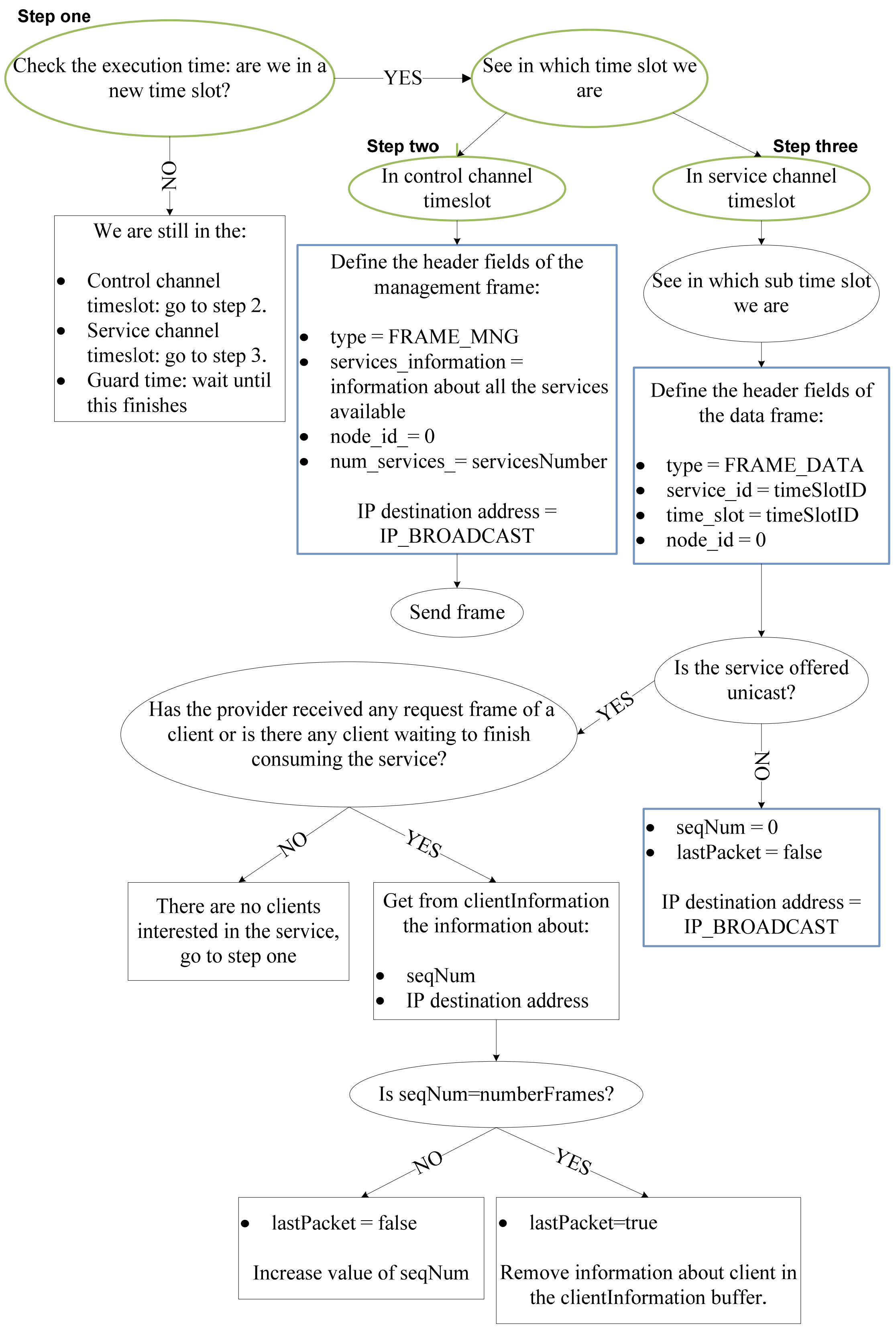

Step one
Check the execution time: are we in a new time slot?
YES
See in which time slot we are
NO
We are still in the:
• Control channel timeslot: go to step 2.
• Service channel timeslot: go to step 3.
• Guard time: wait until this finishes
Step two
In control channel timeslot
Define the header fields of the management frame:
• type = FRAME_MNG
• services_information = information about all the services available
• node_id_ = 0
• num_services_ = servicesNumber
IP destination address = IP_BROADCAST
Send frame
Step three
In service channel timeslot
See in which sub time slot we are
Define the header fields of the data frame:
• type = FRAME_DATA
• service_id = timeSlotID
• time_slot = timeSlotID
• node_id = 0
Is the service offered unicast?
YES
NO
• seqNum = 0
• lastPacket = false
IP destination address = IP_BROADCAST
Has the provider received any request frame of a client or is there any client waiting to finish consuming the service?
NO
YES
There are no clients interested in the service, go to step one
Get from clientInformation the information about:
• seqNum
• IP destination address
Is seqNum=numberFrames?
NO
YES
• lastPacket = false
Increase value of seqNum
• lastPacket=true
Remove information about client in the clientInformation buffer.

Figure 4.4: Diagram of the processes and variables involved in the sendFrame() function from the application layer of the provider protocol stack.

The other important function is *recv()*, which works as shown in Figure 4.5.

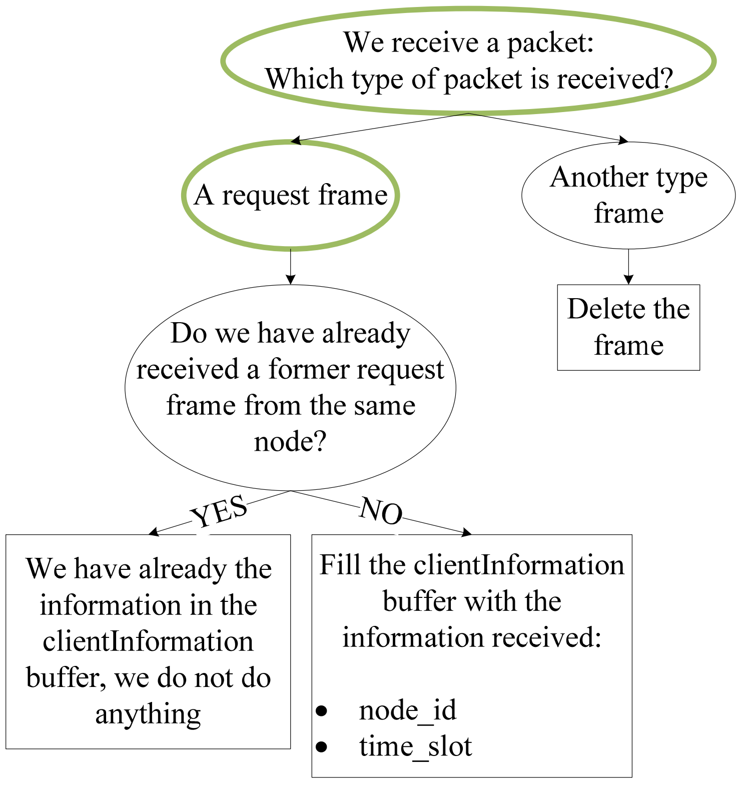

We receive a packet:
Which type of packet is received?
A request frame
Another type frame
Delete the frame
Do we have already received a former request frame from the same node?
YES
NO
We have already the information in the clientInformation buffer, we do not do anything
Fill the clientInformation buffer with the information received:
• node_id
• time_slot

Figure 4.5: Diagram of the processes and variables involved in the recv() function from the application layer of the provider protocol stack.

It is possible to change some of the characteristics of the application layer in the provider side by modifying a group of variables in the Tcl script. These variables are:

- payloadSize: This variable allows us to change the payload of both control and service channel frames. Nowadays all the frames have the same payload but this could be changed (see Chapter 5).
- modulationScheme: Set up the type of modulation used. We can choose between BPSK, QPSK, QAM16 and QAM64.
- servicesNumber: Definition of the maximum number of services, and therefore of subtime slots in the service channel interval, offered by the provider. As mentioned at the end of Chapter 3, we assume this number cannot be bigger than three.

- nodeIdentifier: Set up the identifier of the node. It is an integer value, which will be zero in case of the provider.
- numberUnicastServices: It is also possible to define the number of unicast services that the provider offers. This number cannot be bigger than the value of servicesNumber. The number of broadcast service will be the difference between servicesNumber and numberUnicastServices.
- numberFrames: This variable is used to define the number of frames necessary to consume a unicast service. It is necessary to handle the unicast information between different clients. In our implementation this value is considered to be common to all the unicast services, a fact that could be improved (Chapter 5).

The application in the client side is called **pbc3-sink**. The main function of the application layer in the client side is to receive the frames correctly, sent by the provider, but it is also necessary to have a function which is in charge of sending the request packet when the client is interested in a unicast service. The function which is in charge of receiving the packets is called *recv()* and works as follows:

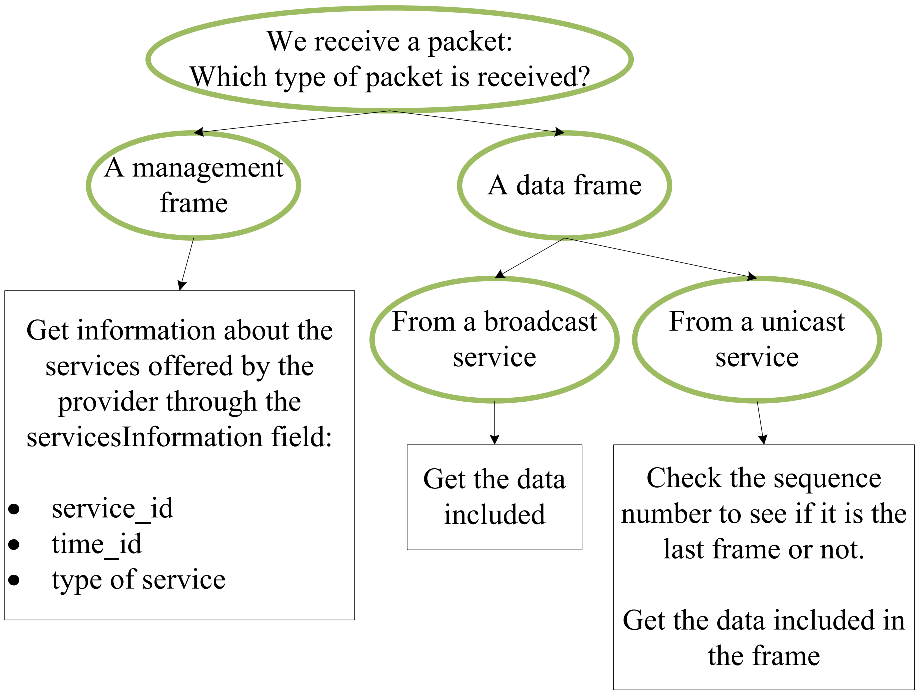

We receive a packet:
Which type of packet is received?
A management frame
A data frame
Get information about the services offered by the provider through the servicesInformation field:
• service_id
• time_id
• type of service
From a broadcast service
From a unicast service
Get the data included
Check the sequence number to see if it is the last frame or not.
Get the data included in the frame

Figure 4.6: Diagram of the processes and variables involved in the recv() function from the application layer of the client protocol stack.

The function in charge of generating and sending request packets is called *sendRequest()* and is defined in Figure 4.7.

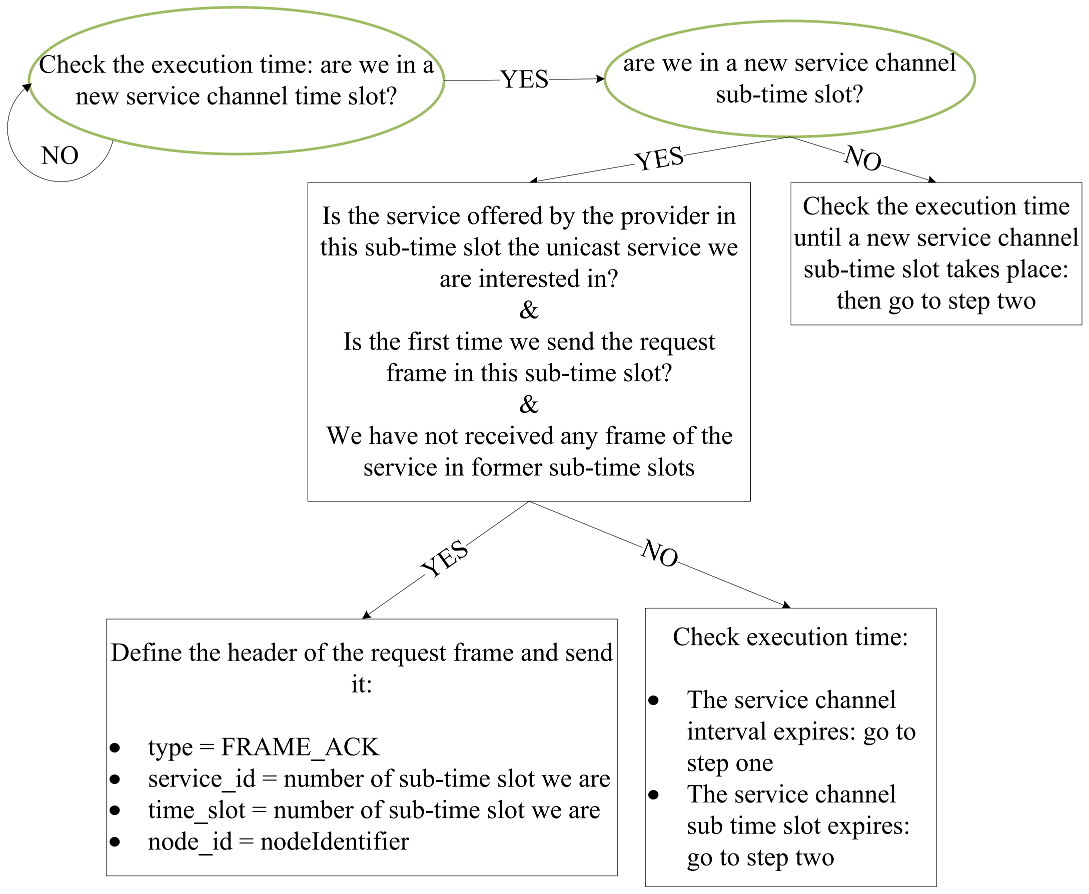

Check the execution time: are we in a new service channel time slot?
NO
YES
are we in a new service channel sub-time slot?
YES
NO
Is the service offered by the provider in this sub-time slot the unicast service we are interested in? & Is the first time we send the request frame in this sub-time slot? & We have not received any frame of the service in former sub-time slots
Check the execution time until a new service channel sub-time slot takes place: then go to step two
YES
NO
Define the header of the request frame and send it:
• type = FRAME_ACK
• service_id = number of sub-time slot we are
• time_slot = number of sub-time slot we are
• node_id = nodeIdentifier
Check execution time:
• The service channel interval expires: go to step one
• The service channel sub time slot expires: go to step two

Figure 4 7: Diagram of the processes and variables involved in the sendRequest() function from the application layer of the client protocol stack.

Also the application layer in the client side can be modified through a Tcl script. There are some variables already used in the application layer of the provider as payloadSize, modulationScheme, servicesNumber and nodeIdentifier and others defined specially for the client side:

- serviceReceived: this variable is used to define which service the client will receive. It is supposed that each client is able to receive one service, this could be improved in newer versions of the protocol.

Once we have explained how our application works in both sides (provider and client) we must explain the main changes done in the MAC layer. When we downloaded the NS version 2.33 there was already included an implementation of the IEEE 802.11a protocol. We did not want to make use of this code because it was totally oriented to guaranty the CSMA/CA with virtual carrier sense mechanism, we were not interested in. There was also a simple TDMA implementation included. We decided to adapt it to our requirements. We basically had to change the definition of the TDMA frame and to set up both data and management MAC headers. In contrast to the application layer there are no variables defined to make use of the MAC layer through the Tcl script.

If we focus on the physical layer, we will see that in our version of the ns2 tool there were already implemented two physical layers for wireless communications: the basic one which is called WirelessPhy and an improvement of it which is called WirelessPhyExt. We were interested in using this last version of the physical channel basically because it introduced an important concept for us: it supports multiple modulation schemes. The WirelessPhyExt allows working with BPSK, QPSK, QAM16 and QAM64 as it is described in [39]. The modulation influences some important characteristics such as the data bit rate and the minimum sensitivity of the receiver, as shown in Figure 3.8, and hence the transmission time of the data and the SINR necessary to receive and decode it. The only problem is that this new version of the wireless channel must be used together with an extension of the MAC layer called Mac802_11Ext. We were not interested in using that one, because for the same reason we were not interested in using the Mac802_11 version. For that reason we decided to introduce the multiple modulation schemes in the WirelessPhy layer.

Another important point when working with the lower layers of the WAVE protocol stack, is to think about how the channel is modelled in the ns2. There are four different types of propagation channels defined and included in the ns2.33: the free space model, the Two-Ray Ground reflection model, the Shadowing model and the Nakagami model. The first three models are well described in [30] (chapter 18) and summarised in the following points:

- Free space model: This model assumes an ideal propagation where there is only one clear line-of sight path between transmitter and receiver. The receive power is obtained from Friis' formula (formula (18.1) in page 188 of [30]).

- Two-Ray Ground reflection model: This model considers two paths, the direct one and the ground reflection. It gives better values of the receive power at long distances, but not in short distances. It requires obtaining a crossover distance, which indicates the threshold distance from where this model can be used. If the distance is smaller than the crossover one, the free space model is used, in order to calculate the receive power. For details see page 189 of [30] and Appendix F.

- Shadowing model: the above two models predicts the mean received power at a specific distance. In reality, the received power is a random variable due to multipath propagation effects, which is also known as fading effects. Those effects are considered in this RF propagation model.

  In this model the receive power is defined by three components:

  $$Power\,\mathrm{Re}ceived\big|_{d} = Power\,\mathrm{Re}ceived\big|_{d0} - 10\beta\log(\frac{d}{d_0}) + X_{dB}$$

  The value β is the path loss exponent and depends on the environment (table 18.1 in [30] gives some typical values of β). The term X_{dB} is a Gaussian random variable with zero mean and standard deviation σ_{dB}. σ_{dB} is called the shadowing deviation, and is also obtained by measurements [30] (table 18.2). The receive power at distance d_0 is obtained by using the Friis' formula.

The Nakagami model is not included in [30], because it was recently introduced in ns2 simulator. Nakagami fading occurs for multipath scattering with relatively large time-delay spreads, in contrast to Rayleigh and Ricean fading models, which are used to simulate small scale fading, the Nakagami model is used to describe large scale experiments on rapid fading in high frequency long distance propagation [40].

Using a specific propagation model only modifies the communication window of each of the OBUs. The communication window is the time that any client or OBU has to communicate with the provider or RSU. The communication window is equivalent to the area of coverage of the RSU. The magnitude of the communication window will strongly depend on the speed of each OBU, the transmitter power of the RSU and the path losses. Since the parameters for the Shadowing and Nakagami model were not clear defined, we decided to use the Two-Ray Ground reflection propagation model for our tests.

5 Problems and Improvements

Until now we have explained the reason why we were interested in studying TDMA technology in C2X Communications and the process followed to define and implement our protocol. When creating a new protocol sometimes it is not possible to develop exactly the theoretical idea we had due to some limitations the simulator has. It is also possible that our implementation could be improved in determined points, we must realise that although sometimes the protocol seems to be complex a lot of improvements could be done to obtain better and more specific results.

The idea of this chapter is just to explain the main problems found when elaborating our protocol and to suggest some improvements we can not carry on because they are out of our scope or we just do not have enough time.

If we refer first to the problems found when working we must clarify that most of them are not really problems (in the sense of bugs found when executing the protocol) but limitations the simulator has which do not allow us to define the protocol as we wanted to. There are three main limitations we want to point out:

The first one is already mentioned in Chapter 4. The problem is related about parallelizing data flows in one node. We were interested in this fact not only for being able to have more than one application running in parallel in the same node (as explained in Chapter 4) but also to define both protocol planes, data and management planes, in the same node (as we can see in Figure 1.1) although this last idea was discarded because it required a lot of work to do.

In chapter four we adopted one easy and fast solution which did not affect to the results obtained. But there are other solutions, the simplest one consists on defining as many nodes, in the Tcl code, as data flows we need and link these nodes through a router. To explain it easily: we will have one node per each data queue (see Figure 3.8) and one router which handles the information form each node. This solution is based in the actual implementation of the Diffserv queues in the NS [41] where virtual and physical queues are used [42]. In our case we will need at least two nodes: one for the data of the control channel of the other for the data of the service channel, in case only one service is offered. We must realise this solution involves changes in the Tcl code which leads to a simple C++ implementation , at least of each node although the router will be more complicated, but to a complex Tcl test implementation.

We decided not to use this solution basically because of two reasons: the first one is this option was also not close to the theoretical idea we had (that was based on the IEEE 1609.4 standard) because it is true that with this solution is possible to have more than one queue of data but they are not defined in the same node; the second one is if we want to define a protocol easy to use for any user we must define all the complex features in the C++ code (which the user will never have to deal with) requirement that the solution explained before does not fulfil. To sum up the user must be able to use our protocol deeply in detail by setting up simple test environments.

Another solution, which could be considered as an improvement of the one we took, is to have only one application running on the provider which generates different types of packets but instead of doing it as a function of the execution time, it could generate them randomly and give the work of organize them to the link layer. In this case we will need to define an algorithm in charge of finding the desired packets in the unique queue that exists in the link layer and sending them in the correct order to the MAC layer.

There is a third solution which allows having an implementation closer to the one specified in the standards ([36] and [43]). The idea consists on adapting the definition of the queue done nowadays in the implementation 802.11e standard which is included in the simulator. As we can see in page 14 of [44] this implementation requires changes in the definition of the class queue which allows having multiples queues by creating them in an array [43]. The source code of this new type of queue can be found in [45].

The second limitation is related to the synchronization of the nodes. Those nodes can be an OBU or a RSU. As it is explained in Chapter 2 the NS is a simulator based in events controlled by timers. The fact is, as it is pointed out in IEEE 1609.4 standard (page 5 from [3]) all the nodes require to be synchronized before communicating between them. The synchronization is especially important when using TDMA technology and it is a process which will be carried on when any OBU enters in the communication area of a new RSU in a centralized system. The fact is that in the NS tool all the nodes implemented (in the Tcl code) have the same time basis which means they do not need any synchronization because they are already synchronized.

If we are interested in defining the synchronization process we should first to desynchronize the nodes by manipulating their timers. In our case we will consider the RSU time basis to be the one the other nodes must to synchronize on. Each OBU will have to follow a synchronization process before receiving data frames from the RSU. The idea could be the following: the first time an OBU receives frames from a new RSU it gets the timestamp of the RSU and, after adding to this timestamp the delay produced by the propagation of the frame, adjusts its timers. Calculate the delay or time difference between the RSU and the OBU is not complicated the only idea which seems not clear is to set up different time basis in the nodes.

The third and last limitation is related to the anti-collisions mechanism used in the MAC layer, which is mainly based in CSMA/CA algorithm. We detected the problem when executing our code: we found there were collisions between request frames when a considered number of OBUs were interested in receiving information about the same unicast service. Those collisions should not take place if we keep in mind each node is supposed to be able of sense the medium to see if it is busy or not before sending any kind of frame.

Why this type of collisions is produced? As we explained in chapter three it is necessary to introduce guard intervals at the end of each time slot to avoid collisions between frames generated by different devices (OBUs and RSU in our case) but in this case the collisions are produced due to different OBUs send the request frame really close in time and, due to they are not able to detect if the medium is idle or not, a collision is produced and detected by the RSU.

This limitation is complicated to solve because it is related to the definition of the communication channel that all nodes use. Although the medium is unique it is instantiated in each node so the nodes can not detect if the medium is idle or not because they have “different” copies of the same thing.

When working with the NS tool we are not able to do all the things we want to not only because of some restrictions or limitations the tool has (as we explained before) but also because we do not have enough time to implement all the ideas which came to our mind and we must simplify and focus our work. Because of this lack of time there exist a lot of points in our protocol which could be improved. Some of these points are explained in the following paragraphs.

If we focus in the implementation of the control channel the most important improvement which could be done is to introduce critical frames and implement the process each node has to follow when receiving those frames. Introducing those types of frames will be really interesting because we can see how to handle both types of information (critical and no critical) and we would make a better use of the control channel than we do in our actual implementation.

If we pay attention to the service channel there are some things which could be improved, the ideas are summarized in the following points:

- In the actual implementation there is not any prioritization between nodes, which means when two or more nodes want to receive the same information (which is unicast) the first who ask for it is the first which receives the data.

  One possibility is to define the priority as a function of the position of each OBU respect the RSU, it sounds coherent to give higher priority to the nodes which are closer to the RSU because their latency (time necessary to consume a service) is smaller. Basically the latency is smaller because the propagation time (one of the terms that defines the latency) decreases if an OBU gets close to the RSU. Anyway this is one possibility which really will not make any difference because the propagation time is despicable in all the OBUs which belong to the same RSU.

- As we explained in Chapter 3 three we need to define the number of frames that the provider needs to send the data of each service. This is necessary to handle the data between different users when it is unicast. In our protocol this information is only defined in unicast services (although could be also used in broadcast services) and the number of frames is the same in all types of unicast services.

  One improvement is the number of frames to consume a service could be different in case the provider is offering more than one service. Another more realistic option (and more complicated) is to define a fragmentation process, in this we will not have to specify the maximum number of frames needed to consume a service but the maximum size of the service payload.

- We have also not considered what happens if the provider has more services to offer than subtime slots available in one frame. As we mention before (see end of Chapter 3) the maximum number of subtime slots (and hence of services) in a service channel timeslot is three in the worst case, when all the services offered are unicast. In case the provider has four services to offer what it could do?

  One option is the provider offers information about all services available even when it needs more than one frame to do it. When it has finished the provider begins offering information of the first service. Another option is, instead of using subtime slots of the following frames to send all the services available, the provider will wait until one service is consumed (and not demanded for another OBU during a specific time) to use the subtime slot to offer the service left. Both options are described in Figure 5.1:

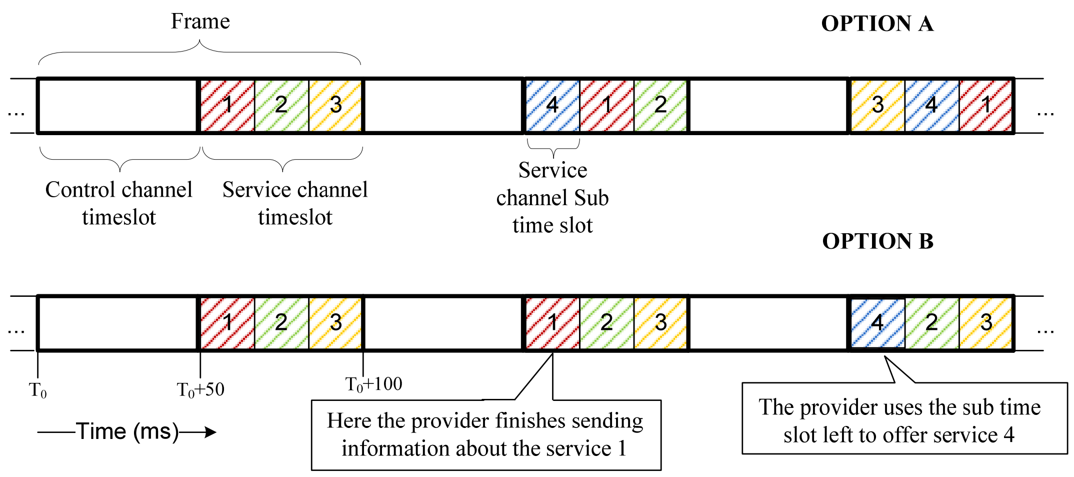

Frame
OPTION A
Control channel timeslot
Service channel timeslot
Service channel Sub time slot
OPTION B
T0
T0+50
T0+100
Time (ms)
Here the provider finishes sending information about the service 1
The provider uses the sub time slot left to offer service 4

Figure 5.1: Possible solutions that can be introduced in the provider side to offer more services than subtime slots available in the service channel.

- Other idea which could be improved is the assignation of each service to each subtime slot in the service channel time slot. Nowadays the provider always uses the first time slots to offer information about the unicast services; the subtime slots left are used for broadcast services. As it is shown by Figure 4.3 any service is defined by an identifier (which is a number), the number of the subtime slot where it is offered and the type of service (unicast or not). In our implementation we decided to make equal the service identifier and the number of subtime slot which means the service whose identifier is one will be always offered in the first subtime slot of the service channel time slot and so on.

  But in fact the identifier of the service and the identifier of the subtime slot should not be the same. The services available in the provider should be distributed in priorities, even when there are enough subtime slots in the service channel. The question is which criteria we must follow to assign a priority to each service. To have criteria requires the services to be defined more in detail, idea which is not implemented in our protocol.

A good idea could be to define the rules to classify the services into different priorities and study which could be the best assignation of the subtime slots to the services in sense of improving the latency.

We only considered the losses of frames produced by collisions but there is also the possibility of having losses due to the propagation channel has a probability of error associated which means that a percentage of the frames sent can not be used due to an error. In the NS simulator it is included an error model which introduces packet losses into a simulation but this error model is not associated to the radio propagation model. A good idea could be to modify the propagation model by introducing a probability of error associated. As this error will affect any frame or packet sent (from both service and control channel) and therefore makes necessary to introduce a bidirectional communication between provider and client in both intervals (remember that our protocol only considers a bidirectional communication when transmitting unicast service messages) because the provider must verify that all packets sent are received by the client.

6 Results

The last step, once we implemented and debugged our MAC protocol, is to obtain results, which allow us to compare both TDMA and FDMA technologies. In general when we want to compare two things we must compare them in the same conditions, this means we should compare both technologies by using the same WAVE device and the same environment (which basically means the same propagation environment). But under which conditions we can assume that both TDMA and FDMA are comparable? The following points sum up the most important ones:

- The FDMA WAVE device works nowadays with three channels; this means the RSU would be able to transmit two service channels at the same time, as we can see in Figure 1.6. Our TDMA WAVE device (and in general all the TDMA WAVE devices with similar characteristics) is able to work only with one channel per time so if we want the provider to offer two different services we should divide the service channel interval into two subtime slots. We can state that in this case both technologies are similar in sense of the information given.

- A single transceiver is assumed to be used in both cases, which means, in case of FDMA, only one channel can be detected in each communication interval.

- Although the TDMA based implementation allows using channels larger than 10 MHz we will maintain the same channel bandwidth used in the actual FDMA WAVE devices.

To obtain all the results we show in this chapter at first how we set up a Tcl script (see Appendix A), following the steps explained in Chapter 2 and using the variables specified in Chapter 4. An important point we have not paid enough attention is the definition of the movement of the OBUs. Although the behaviour of our protocol does not depend on the position of the OBUs it is important to have a realistic definition of the movement because it will strongly modify the communication window. Those movements could come from a traffic simulator (see Chapter 2) but we decided to define them as it is explained in [30] (page 160). We got the movement traces from Manuel Zaera's master thesis [46]. The movement of OBUs and their initial position respect to the RSU are shown in Figure 6.1.

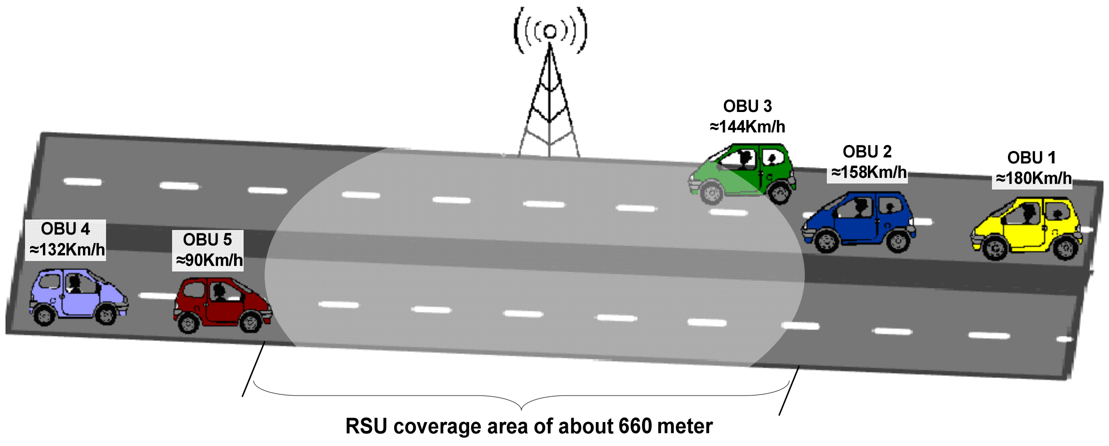


Figure 6.1: Simple scheme which shows the relative position and speed of each OBU with respect to the RSU, based on [46].

All our simulations will be based in the scenario shown in Figure 6.1 which is a simplification of the scenario used in [46]. The movement traces correspond to a real data obtained from a german highway for a specific period of time which means that when the measures were taken the OBU 1 was about overtaken the OBU 2. Anyways our simulation tests last only one minute and hence the OBU 1 has not enough time to overtake OBU 2 so both of them maintain their relative positions during all the simulation. The same fact takes place between OBU 4 and OBU 5.

Once we have defined our Tcl script we should think about which parameters allow us to analyse and compare efficiently both technologies. Obviously these parameters should be strongly dependant on the multiplex technique. What could be an interesting parameter? We could think about the channel interference, reason why we decided to study TDMA, but it is complicated to compare just because there are no measures of these interferences (in sense of losses or degradation of the transmitted signal FDMA based devices) and in case of TDMA there are not results just because they do not take place.

Another important parameter is the latency or time necessary to consume the service, which let us study which technology is the best in sense of speed. All the figures below this paragraph show different measures of the latency for different situations which we considered interesting to analyse. The situation we were more interested was how the existence of more than one user influences the latency when a unicast service is offered. To analyse this situation we defined three cases depending on the number of unicast services offered by the provider:

- **Case A**: There is only one unicast service offered by the provider. Therefore the provider will have 44 ms available to send information about the service:

  DurationSCHsubtimeslot-TimeToReceiveRequestFrames-TimeGuard →
  50 ms – 3 ms – 3 ms = 44 ms

  This means the provider can attend three users per SCH interval, because the unicast service requires five frames (each one of 1000 bytes) to be consumed:

  5 frames / Service data x 2.712 ms / frame ≈ 14 ms / Service data →
  44 ms / 14 ms ≈ 3.1 clients / SCH interval

- **Case B**: There are two unicast services offered by the provider, so the time available is 19 ms and therefore only one user can be attended per SCH interval. During the 5 ms left (19 ms – 14 ms) the provider will begin sending service data frames to another client in case there is more than one client interested in the same service (see Figure 3.3).

- **Case C**: There are three unicast services offered by the provider and hence the duration of each service channel subtime slot is about 11 ms. In this case there is not enough time to attend one client at all (14 ms are required) so we will appreciate a considerably increase of the latency.

For the three cases of study we will execute the same test environment (the only difference will be the number of unicast services offered by the provider) and we will obtain two types of figures. The first one will show the average time that a specific OBU needs to receive all the data of the service depending on the number of clients (the minimum will be one and the maximum five, see Figure 6.1) that are interested in receiving the service. This average time will be measured in three ways:

- **Measure 1**: We define the latency as the time since the provider sends the first frame until the client receives the last one (being more specific, time necessary to send the five frames which define the service).

- **Measure 2**: In this case the latency is defined as the time since the client sends the last request frame until it receives the last data frame of the service. We must remember that if the client is not attended in one SCH interval it will send again the request frame in the following service subtime slot. Measure two only considers just this last request frame.

- **Measure 3**: The latency is defined as the time since the first request frame is sent since the last frame is received by the client. In this case we take into account the situation where the provider is not able to attend a client (because there is not enough time) and this forces the client to send another request frame in the next SCH interval.

To understand better the meaning of Measure 1, Measure 2 and Measure 3 we show an example in Figure 6.2:

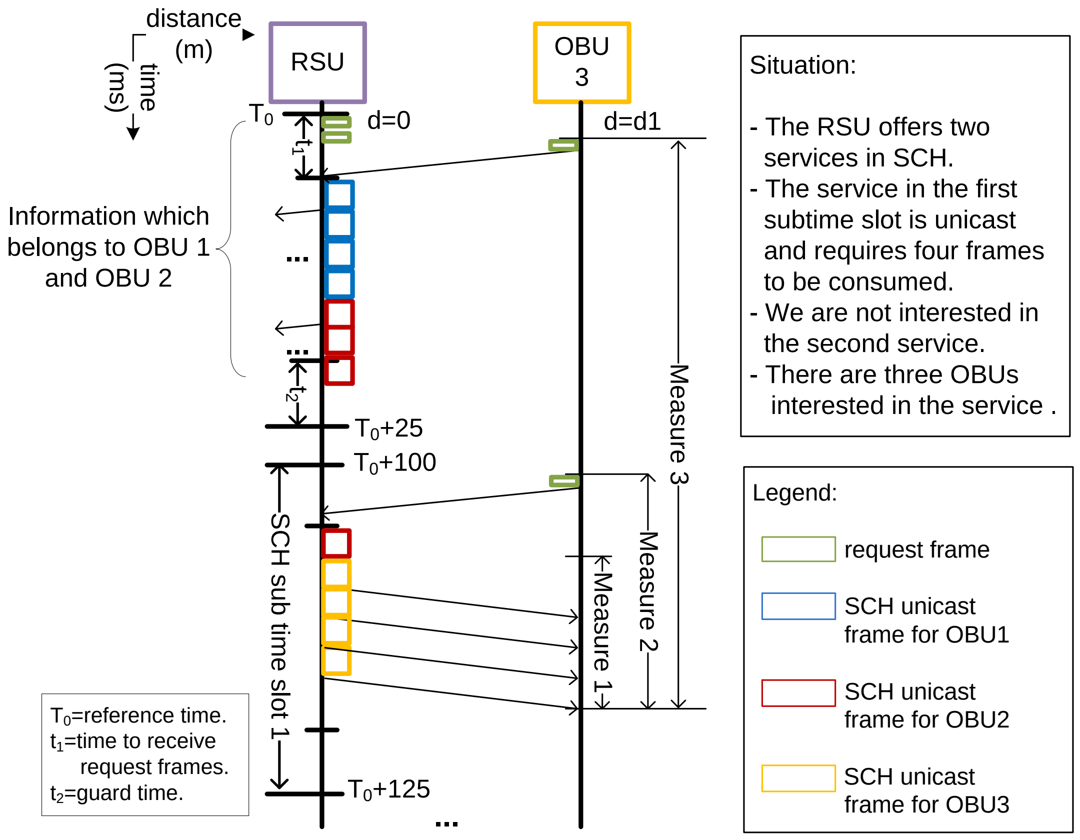

distance (m)
time (ms)
RSU
OBU 3
T0
t1
d=0
d=d1
Information which belongs to OBU 1 and OBU 2
t2
T0+25
T0+100
SCH sub time slot 1
T0+125
Measure 1
Measure 2
Measure 3
T0=reference time.
t1=time to receive request frames.
t2=guard time.
Situation:
- The RSU offers two services in SCH.
- The service in the first subtime slot is unicast and requires four frames to be consumed.
- We are not interested in the second service.
- There are three OBUs interested in the service .
Legend:
request frame
SCH unicast frame for OBU1
SCH unicast frame for OBU2
SCH unicast frame for OBU3

Figure 6.2: Different ways to measure the latency of OBU 3.

The second picture will show us the minimum, the average and the maximum time necessary to consume the service in case we use the Measure 3 for a specific OBU. To obtain these values we run the Tcl script (whose main characteristics are shown in Appendix A) and from the trace file obtained (Appendix B) we calculate the latency for Measure 3 (and write it in a text file) by using a Perl script (Appendix C). We repeat this process 100 times so we have 100 latency measures in a text file. The average latency is just the average of these latency measures and the minimum and maximum latency are the minimum and maximum value obtained in the text file respectively. We decided to show the minimum and maximum latency because we realised sometimes the average time does not reflect all the characteristics we are interested in.

We must point out all the latency measurements are based on OBU 5. We chose to base our analysis in this OBU, because it is the slowest OBU (and therefore the OBU with the largest communication window) and the first OBU in communication with the provider (RSU).

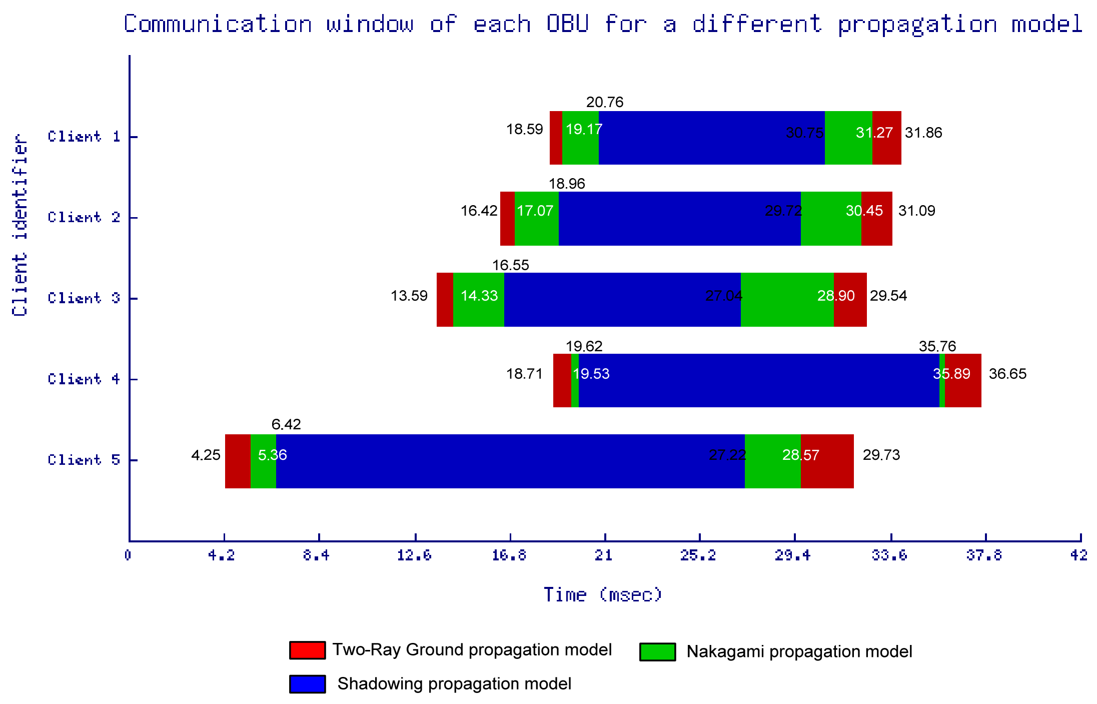


Figure 6.3: Communication window of each client

Figure 6.3 shows the communication window of each OBU (or client) for the three propagation models explained in Chapter 4. Anyway all the results consider the case where a Two-Ray Ground propagation model is used.

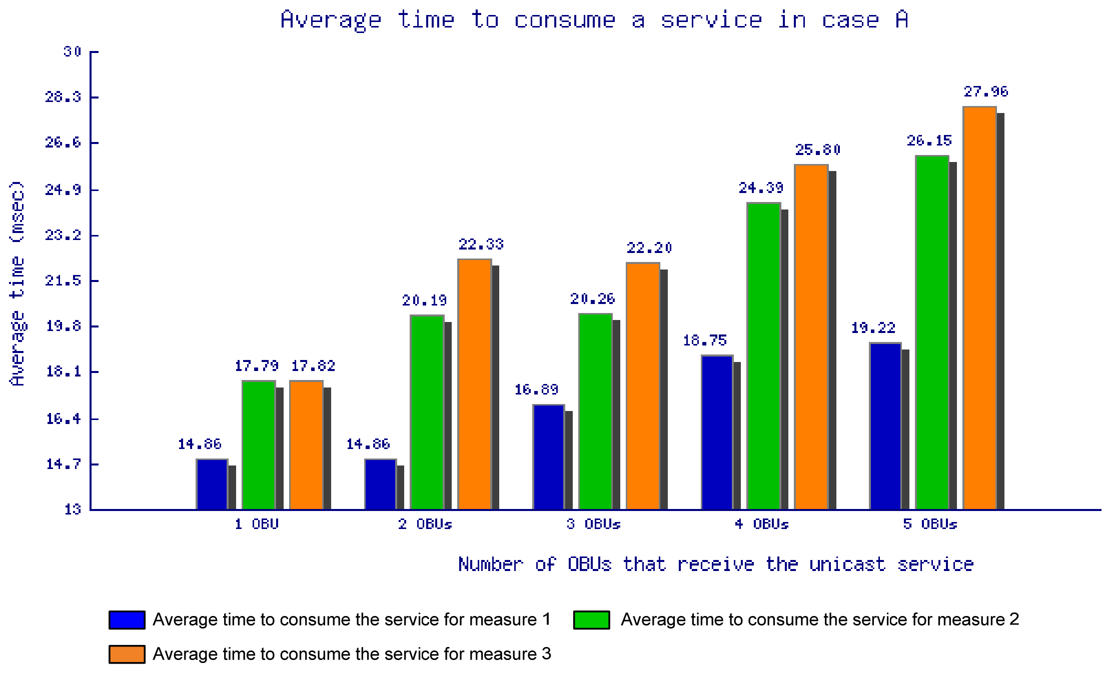


Figure 6.4: Average latency for OBU 5 when one unicast service is offered by the provider.

In Figure 6.4, 6.6 and 6.8 we are interested in appreciating how the number of OBUs that receive the unicast service affects the latency of one specific OBU (in our case the one whose identifier is 5). The aim of these figures is to see the relation between the bars which show different measures of the latency or between bars obtained for a different number of clients.

It is important to realise that the values obtained for the three types of measures are an average of multiple latency measurements done (the process is explained in page 64) and therefore sometimes it could happen that the magnitude of the values does not correspond to a theoretical value. That is the main reason why we decided to show the minimum and maximum value of the latency for OBU 5 when using Measure 3 (Figure 6.5, 6.7 and 6.9).

In Figure 6.4 it is interesting to see how the bars green and orange measure the same value in case the OBU 5 is the only one communicating with the RSU, this is because although the measurement of the latency is different is this case they are equivalent (pay attention to Figure 6.2). Also we point out that the bars for Measure 2 and 3 have almost the same value when there are two and three OBUs receiving the data. This means that for OBU 5 sharing the service channel with one OBU is the same (in terms of average latency) than sharing it with two OBUs, in case there is only one service offered by the RSU.

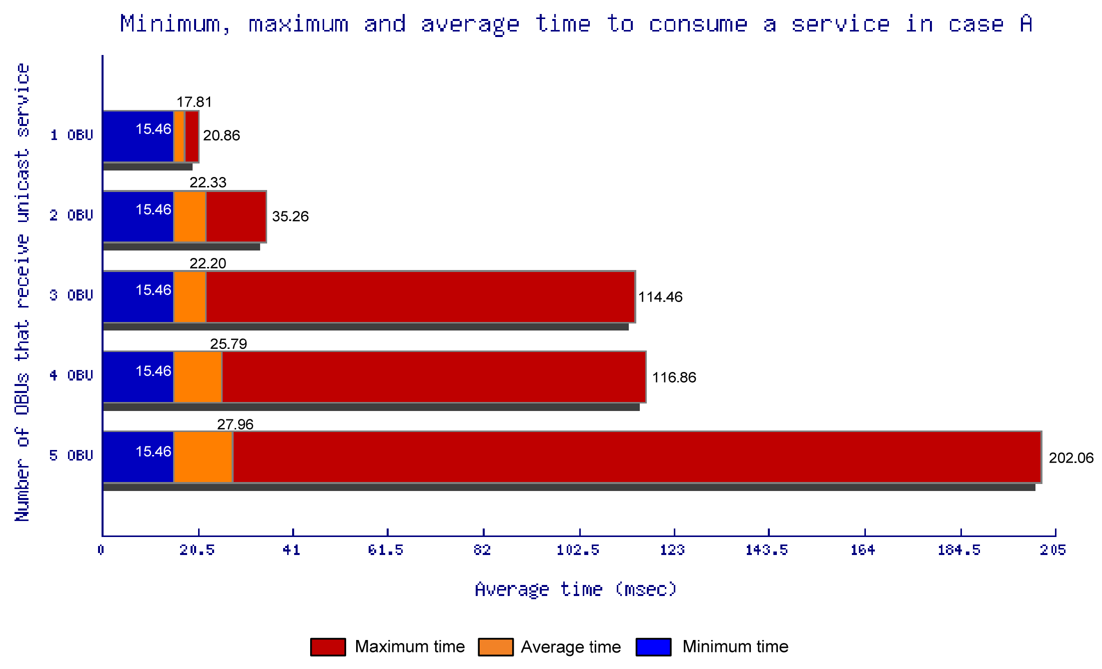


Figure 6.5: Relation between the minimum, maximum and average latency that OBU 5 needs to consume a service when only one unicast service is offered by the provider.

In Figure 6.5 we can appreciate how the minimum latency (of the OBU 5) is independent of the number of OBUs that receive the unicast service. This minimum latency is produced when the request frame of OBU 5 is the first one received by the provider and therefore the OBU 5 is the first one to be attended. This characteristic can be also seen in Figure 6.7 and 6.9. If we pay attention to the maximum latency, which constitutes the worst case in sense of consuming the service we can see how it increases considerably with the number of OBUs. We can compare the results obtained with the simulator with the ones theoretical:

$$1\ OBU: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 0ms + 3ms + (2.712ms \times 5) \approx 17ms \quad (6.1)$$

$$2\ OBUs: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 0ms + 3ms + (2.712ms \times 10) \approx 31ms \quad (6.2)$$

$$3\ OBUs: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 100ms + 3ms + (2.712ms \times 1) \approx 106ms \quad (6.3)$$

$$4\ OBUs: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 100ms + 3ms + (2.712ms \times 5) \approx 117ms \quad (6.4)$$

$$5\ OBUs: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 100ms + 3ms + (2.712ms \times 10) \approx 131ms \quad (6.5)$$

When comparing both results (the theoretical ones and the simulation ones) it is common that the simulation results are bigger than the theoretical ones. This is due to the formulas used to calculate the theoretical results are approximated. In the formulas we always assume that there are not time losses but in fact there are situations where time is lost:

- We assume the provider begins sending data frames just once the time destined to receive acknowledgement frames finishes, in fact, it could happen the provider spends some milliseconds before it begins sending data frames.

- The same fact occurs when the provider finishes sending data frames to a user and therefore begins sending frames to another user in the same service channel timeslot.

Which is the value of this time wasted? As we described in Chapter 2 the NS is an event simulator: an event is created, executed and finished. This process is repeated until the simulation time finishes. The existence of events allows having a time-based simulation. In our case we decided that the duration of each event will be the time necessary to send a data frame (2.712 ms), and therefore the maximum time wasted corresponds to an event. Obviously these losses do not take place always, this makes complicated to obtain a theoretical result coincident with the simulation one.

If we pay attention to the results obtained when all the clients are interested in the same service (five clients) in Figure 6.5 we appreciate a considerably difference between the simulation result (202.06 ms) and the theoretical value (formula (6.5)), in this case if we include the time losses explained before in formula (6.5) we will obtain a more accurate theoretical value:

$$5\ \ OBUs: latency = 2t_{frame} + t_{request} + t_{losses} + t_{txData} = 200ms + 3ms + 3ms + 2.712ms \approx 209ms \quad (6.6)$$

In Appendix E it is explained in detail how the formulas (6.5) and (6.6) are calculated. The process shown in Appendix E is applicable to all formulas used in this chapter.

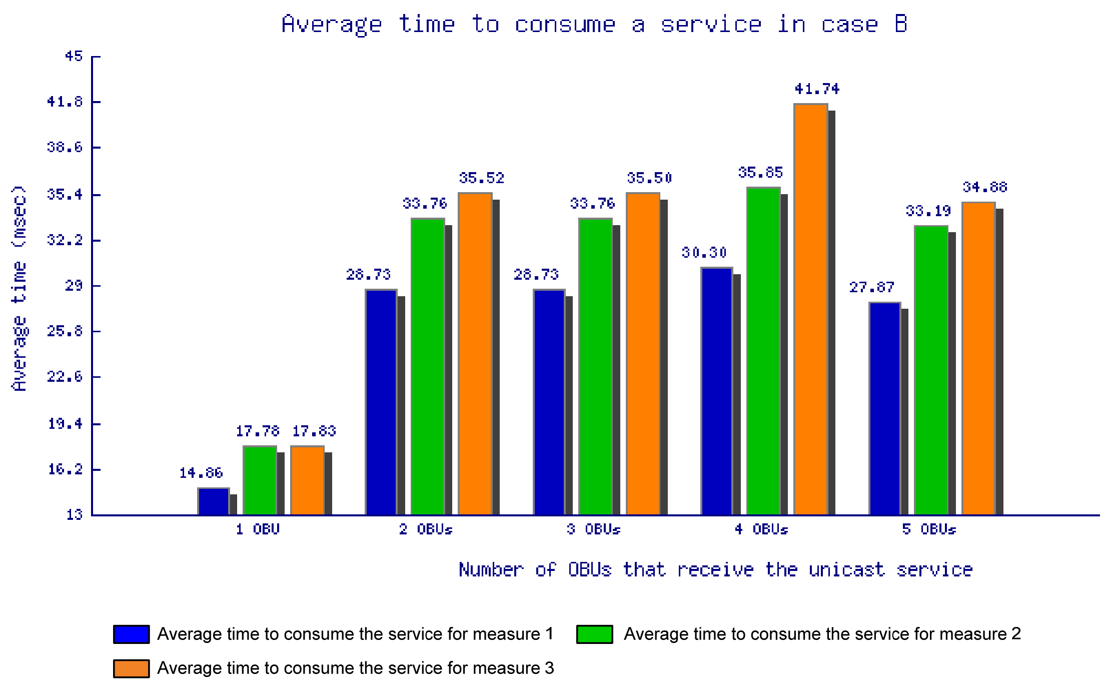


Figure 6.6: Average latency for OBU 5 in case two unicast services are offered by the provider.

Comparing Figure 6.4 and 6.6 we can see how the values are similar in case only one OBU is receiving the data. In contrast we can appreciate how the latency (for any type of measure) increases considerably when more than one OBU wants to receive the data. This is due to the subtime slot of the service is reduced to the half when the provider has two different services to offer.

In Figure 6.6 there is also another fact important to analyse. We can see how in case all clients (in total five) are interested in the service, the average time (or average latency) that the OBU 5 needs to consume the service for any type of measure is smaller that the average time necessary when there are four OBUs demanding the same service. Although this does not seem correct there is one reason why this could occur:

- Do not forget the values shown in Figure 6.6 (and in Figure 6.4 and 6.8) are an average of multiples results obtained. Having a small average latency when five clients demand the service than when four clients demand it only means that the probability of OBU 5 to be the last one to send the request frame is smaller in case all clients are demanding the service than in case only four clients are demanding it.

We can appreciate this fact if we pay attention to Figure 6.7. In this figure we see how the maximum latency for OBU 5 when all OBUs demand the service is 308.26 ms value that can only be obtained when OBU 5 is the last one to consume the service (see formula (6.11)). This means the OBU 5 is sometimes the last one to consume the service when all clients demand it, and hence, the reason why the average latency decreases is because the probability of the OBU 5 being the last one to consume the service in this case is smaller that when four OBUs are demanding the service.

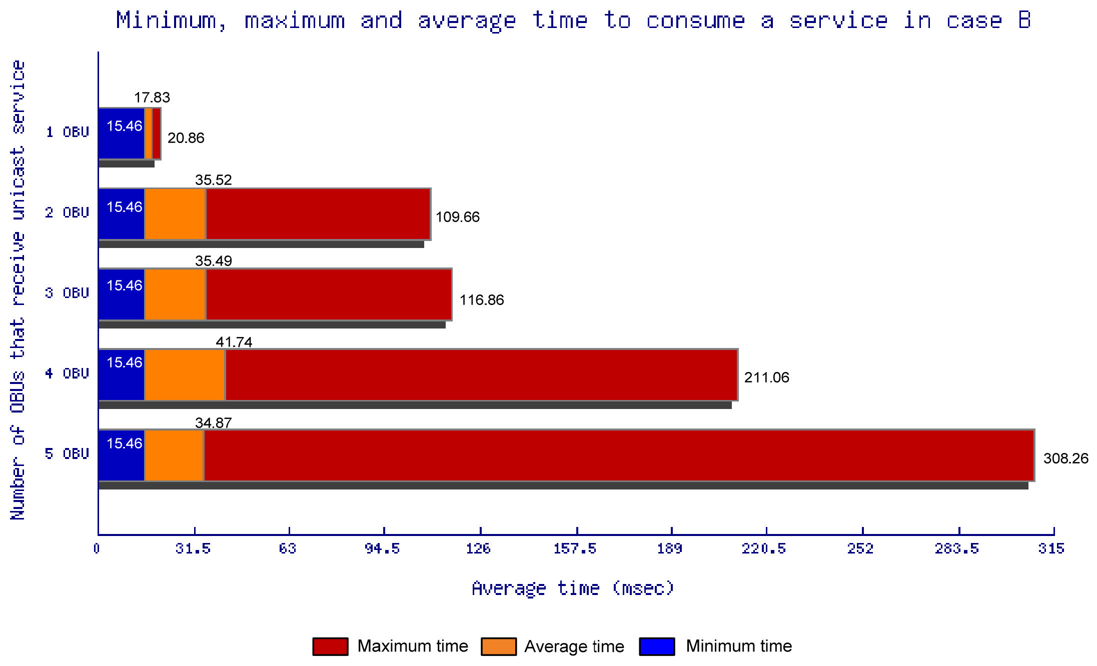


Figure 6.7: Relation between the minimum, maximum and average latency that OBU 5 needs to consume a service when two unicast services are offered by the provider.

As we did with Figure 6.5 we are going to pay attention to the worst case: when the latency is maximum. The theoretical maximum latency values obtained for case of analysis B are:

$$1\ OBU\ : latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 0ms + 3ms + (2.712ms \times 5) \approx 17ms \quad (6.7)$$

$$2\ OBUs : latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 100ms + 3ms + (2.712ms \times 3) \approx 112ms \quad (6.8)$$

$$3\ OBUs : latency = 2t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 200ms + 3ms + (2.712ms \times 1) \approx 206ms \quad (6.9)$$

$$4\ \ OBUs: latency = 2t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 200ms + 3ms + (2.712ms \times 6) \approx 219ms \quad (6.10)$$

$$5\ \ OBUs: latency = 3t_{frame} + t_{request} + (t_{txData} \times N^{o}{}_{frames}) = 300ms + 3ms + (2.712ms \times 4) \approx 316ms \quad (6.11)$$

In this case, we appreciate a big difference of time when there are three OBUs interested in the same service: the theoretical maximum latency is about 206 ms (formula (6.9)) while the simulation result is about 116.86 ms. although this does not seem coherent we are going to explain the main reasons why this could happen:

- The maximum latency of OBU 5 (remember we base all our analysis in this OBU) is obtained when this OBU is the last one to send the acknowledgement frame to the provider, this means that if there are three clients interested in the service the maximum latency is obtained when the request frame of OBU 5 is the third request frame received by the provider and so on.

  Anyways being the last one to send the request frame does not take place always (the access to the medium is random). In this case we found out that when three clients were interested in receiving the same unicast service (for case of analysis B) the OBU 5 did never send the request frame in the third position and therefore the maximum latency obtained when three OBUs demand the service is similar to the one obtained when only two OBUs demand it. This could also be appreciated in Figure 6.6 where we can see how the bars obtained when two and three OBUs are interested in the service have the same value.

  The reason why it could happen that the OBU 5 is never the last one to send the request frame is because of the timer associated to this OBU and therefore is an intrinsic characteristic of the simulator used.

- It is also important to keep in mind that all the theoretical results consider the situation where all the OBUs send the request frame in the same service channel timeslot. But this is not always true. In Figure 6.3 we could see that each OBU begins to communicate with the RSU in a different time (their communication window is different) this makes the probability of all OBUs send the request frame in the same service channel timeslot to be small. What it is common to take place is that there is already a client consuming the service when OBU 5 sends the request frame especially when the number of OBUs demanding the service is higher. If there is already a client consuming the service when OBU 5 decides to send the request frame to the provider the latency obtained in the simulation will be smaller than the theoretical one.

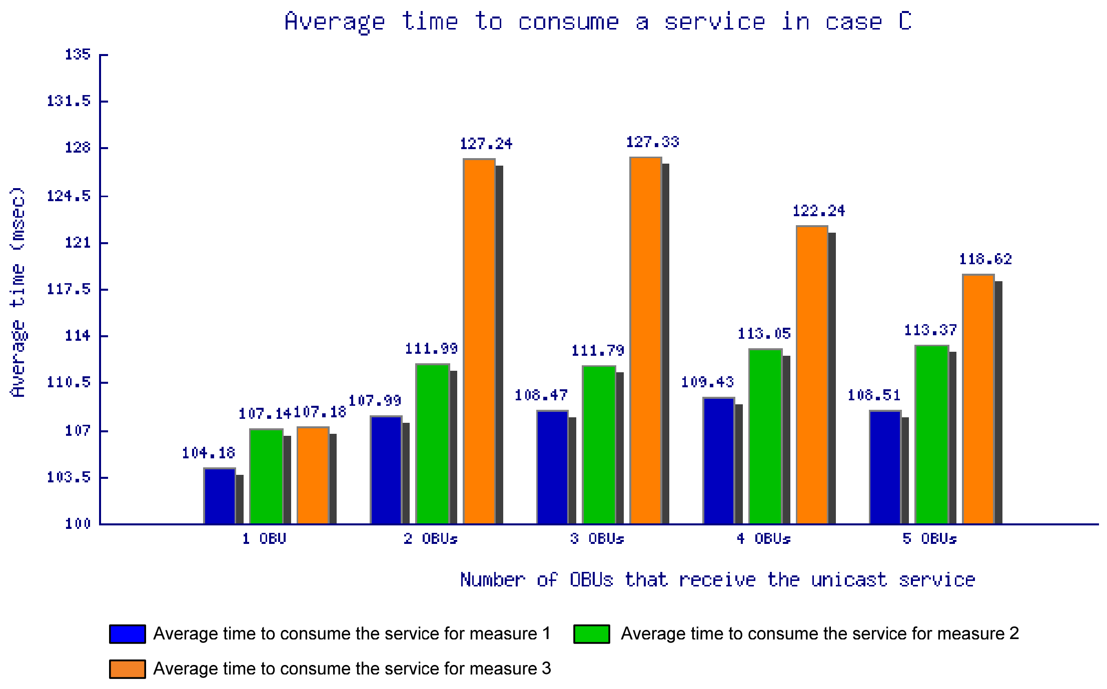


Figure 6.8: Average latency for OBU 5 in case three unicast services are offered by the provider.

In Figure 6.8 we have the same problem that in Figure 6.6: the average latency of OBU 5 Measure 3 is smaller for the cases where four and five clients demand the services. The reason why this happens is because the probability that OBU 5 is the last one to send the request frame when there are four or five clients interested in the services is really small and hence when calculating the average time the bars are smaller comparing them with the bars for the case where there are only two or three clients interested in the service.

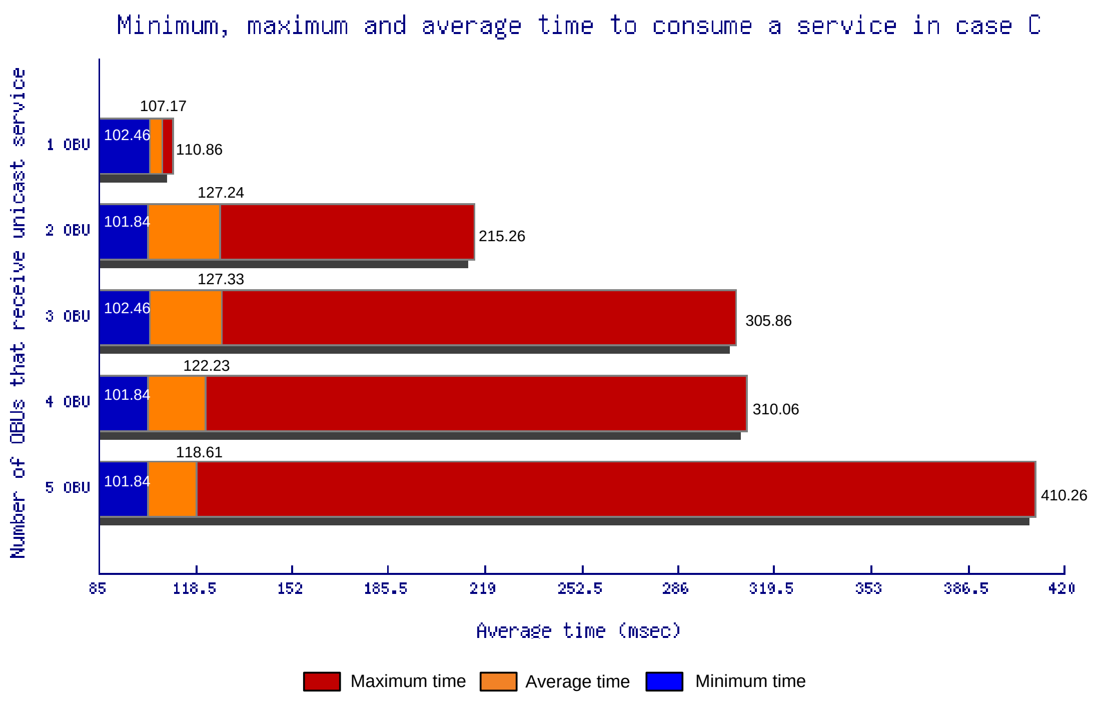


Figure 6.9: Relation between the minimum, maximum and average latency that OBU 5 needs to consume a service when three unicast services are offered by the provider.

In case the SCH interval is divided into three subtime slots (because the provider wants to offer three services) the latency suffers a big increase due to there is not enough time so attend one user in one service subtime slot (remember five frames are required to be sent to consider the whole data of the service is consumed). We point out how even when there is only one client receiving the data it is required more than one frame to consume the service (the latency for any type of measure is bigger than 100 ms).

He can compare the values obtained in Figure 6.9 with the theoretical values:

$$1\ \ OBU: latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}_{\ frames}) = 100ms + 3ms + (2.712ms \times 1) \approx 106ms \quad (6.12)$$

$$2\ \ OBUs: latency = 2t_{frame} + t_{request} + (t_{txData} \times N^{o}_{\ frames}) = 200ms + 3ms + (2.712ms \times 2) \approx 209ms \quad (6.13)$$

$$3\ \ OBUs: latency = 3t_{frame} + t_{request} + (t_{txData} \times N^{o}_{\ frames}) = 300ms + 3ms + (2.712ms \times 3) \approx 312ms \quad (6.14)$$

$$4\ \ OBUs: latency = 4t_{frame} + t_{request} + (t_{txData} \times N^{o}_{\ frames}) = 400ms + 3ms + (2.712ms \times 4) \approx 414ms \quad (6.15)$$

$$5\ \ OBUs: latency = 6t_{frame} + t_{request} + (t_{txData} \times N^{o}_{\ frames}) = 600ms + 3ms + (2.712ms \times 1) \approx 606ms \quad (6.16)$$

There is a considerably difference between simulation and theoretical results when four and five OBUs demand the service, we can see how in both cases the simulation results are smaller than the theoretical ones. The reason why this fact takes place is that when there are four OBUs demanding the service the OBU 5 will never be the forth to send the request frame (the simulation result is 310.06 ms almost identical to the maximum latency obtained when three clients demand the service), in case all clients are demanding the service the OBU 5 will never be the fifth one to send the request frame; the maximum latency obtained in the simulation is 410.20 ms which means the OBU 5 sends the request frame in the fourth position. This is the same fact that happens in case of analysis B.

Finally in Figure 6.10 we show the average time that each OBUs needs to receive the data in different intervals of time of the simulation. We can appreciate how, as the number of clients that get to communicate with the RSU increases the average latency gets high. The aim of this figure is just to show us the evolution of the latency (average value) in time for each of the OBUs. In Figure 6.10 it is also possible to appreciate the communication window of the OBUs.

As it was said before, all the values shown in the figures of this chapter are obtained from reading the traces files (that are generated after executing the TCL script) with a Perl script [29]. In the Appendix C it is included one of the Perl scripts done to read those trace files. To get more information about how to use Perl scripts to read trace files consult [28] and [46]. Once the trace files are read, the pictures are generated from the data obtained by using another type of Perl script (which makes use of the GD:graph tool), the Appendix D show us an example of these Perl scripts.

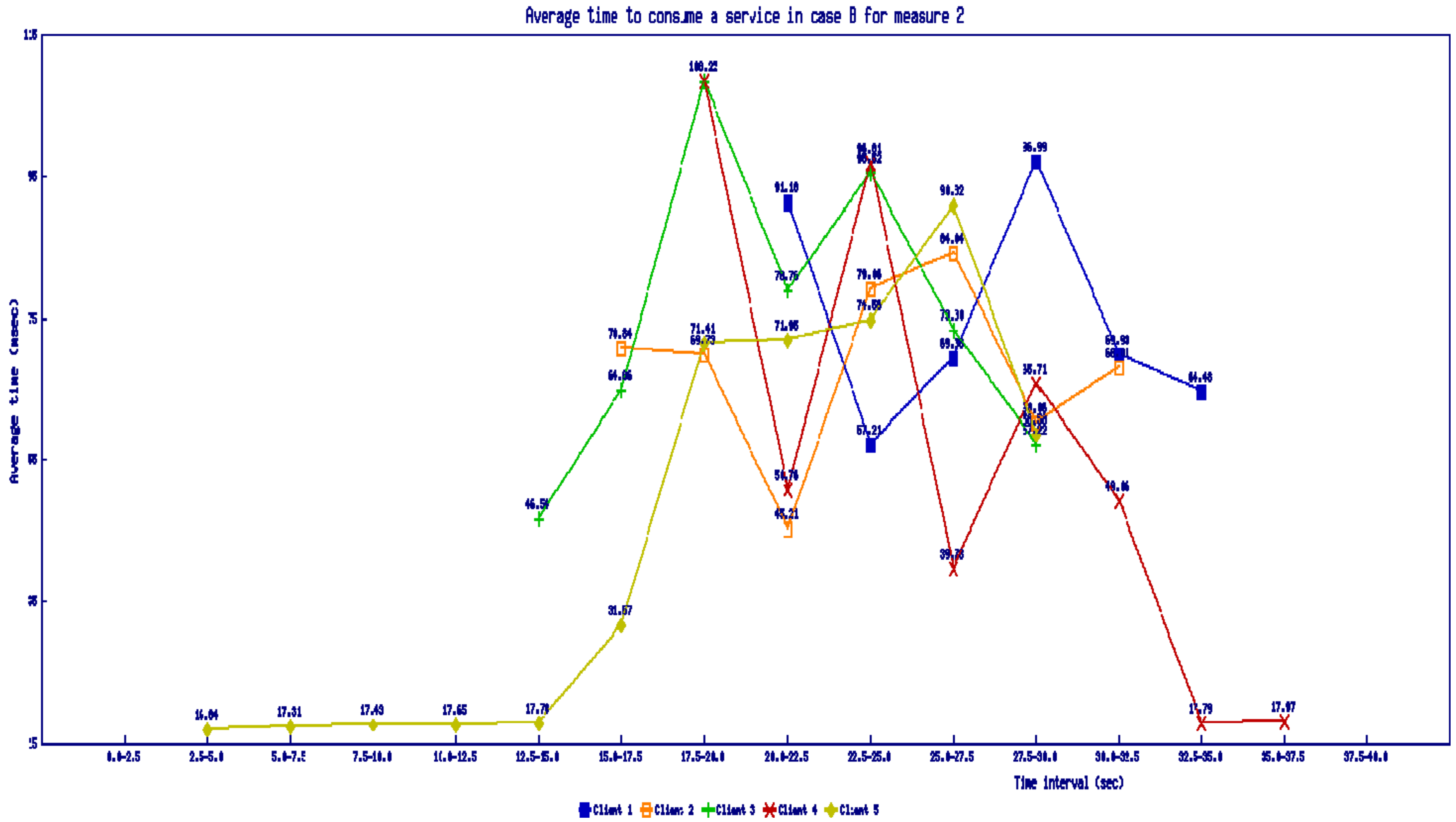


Figure 6.10: Evolution of the average time necessary to consume the service for each OBU when considering the simulation time divided in intervals.

7 Conclusions

All the work done in this diploma has a goal: study if TDMA technology is better than the actual FDMA based implementation. The idea of using TDMA instead of FDMA was to avoid the interference between channels that nowadays take place when using FDMA. Our TDMA based implementation does not have these interferences just because only one communication channel is used (instead of the three channels in FDMA case in Europe) and this constitutes one of the main advantages of our protocol. But having only one communication channel has also some disadvantages, the most important in the latency. We define, in a general way, the latency as the time the client needs to consume the service or, just the way around, the time the provider requires to send all the data of a specific service to a user. To study better the pros and the cons of each technology in sense of latency we are going to analyse them for a specific configuration of the service channel interval.

As we described in Chapter 1 the actual FDMA implementation handles two service channels set up in different frequencies, this means the provider is able to offer two different services at the same time in the same service channel interval. For being able to offer two different services in a TDMA based WAVE device we should divide the service channel interval into two subtime slots: the provider will use the first one to offer the Service A and the second one to offer the Service B. This means that, while in case of FDMA any user will have the whole service channel interval to receive information about the service requested. In TDMA any user will have half of the time to receive the same information. In terms of latency in case of using TDMA access any user will require the double time to consume the service than in FDMA case. This really seems a big disadvantage because it makes TDMA to be slower.

Anyways if we think it carefully we will realise this is not so big problem as it seems to be. First of all the latency is larger in TDMA case only with information related to the service channels because it is the only situation where there is the possibility of different users demanding different services. In case of the control channel interval, FDMA and TDMA will require the same latency just because during this interval all the users will receive the same information (that is the main reason why two communication channels are defined). Also we cannot forget that we decided to base our study in channels of 10 MHz which means that there are 20 MHz of bandwidth not used in TDMA scheme. A good idea could be to use a channel of 20 MHz to compensate this big latency obtained. In this case although the time destined to each service in TDMA implementation is just the half of the time that in FDMA case we will obtain the same latency because the bandwidth is the former case is the double.

Thirdly TDMA is a scheduled based protocol which means the access to the medium is rigid in sense any client cannot send the information wherever it wants to, but in a specific time. This fact could seem a disadvantage in terms of latency because a specific information must be sent in specific time of the frame when, in case of FDMA, this does not happen. But the actual FDMA based implementation makes use of CSMA/CA which guaranties the non existence of collision by using what it is called backoff time in case the medium is not idle. If the number of clients demanding the same unicast information is high the time lost in the backoff process will affect considerably the latency and therefore the results obtained. We can see an example of this in the work done by Manuel Zaera [46], if we pay attention to his Figure 5.3 we will see how the time necessary to access the medium and therefore to send the information is about 0.75 minutes (45 sec) when six mobile nodes (OBUs) are trying to communicate with the RSU. This means that, in the worst case, an OBU needs almost 45 sec to get a free communication with the RSU, in this situation we can appreciate that although TDMA seems to be slowly sometimes we will also have high latencies when using a FDMA-CSMA based system. Obviously the usage of CSMA/CA is interesting when the numbers of devices (RSU or OBUs) that want to access to the medium is small.

Being a scheduled based protocol makes TDMA to waste resources when the demand of a service is small: What happens if there is no client interested in a service offered? In case of FDMA there will be no problem: the clients will continue tuning the control channel, but in case of TDMA this time will be lost, because any client will make use of the information offered. Anyway probably this problem does not take place in real life because, from an economic point of view, does not seem coherent to set up a system where we are wasting time (and therefore bandwidth) doing nothing. We must realise that we did not defined the characteristics of the services offered in the service channel interval this is an important point if we want our wave device to be attractive to any user.

Although our work is deeply based in how the multiplexing affects the different services offered in the service interval we cannot forget the main aim of a WAVE device is to guaranty that safety messages are received properly. These safety messages are sent during the control interval (using the control channel). The usage of the control channel is the same in both access technologies: only one communication channel is transmitted and the information is broadcasted to all the clients. As it is said in IEEE 1609.4 standard the information given in the control channel interval cannot be fragmented and therefore it is guaranteed that if an emergency appears during this interval all the clients will receive it.

When talking about the control channel interval the only difference between TDMA and FDMA takes place when an emergency (that requires a safety message) appears during the service channel interval. In case of using FDMA technology this safety message will be received by the clients that, being not interested in any service offered by the provider, decide to continue tuning the control channel during the service channel interval. In contrast, when using TDMA, any client will not be able to receive this information until a new control channel interval takes place. To sum up not receiving safety messages during the service channel interval is a problem that affects all the clients in a TDMA based WAVE device while in case of a FDMA based WAVE device is a problem which affects only to a part of these clients.

Anyways, is it really a big problem not to receive a safety message during the service channel interval? Or saying it the way around, is it a problem to have a delay of about 50 ms (service interval duration) when receiving safety messages? Probably not. If we keep in mind that nowadays the objective of a WAVE device is to show these critic information in a display and therefore the driver is the one who has to make use of this information in the last term having a delay of 50 ms does not constitute any problem because the driver is not able to assimilate the information so fast. The same idea can be applied when talking about latencies in case of the service channel information, although the faster the better we should take into account we are working with frames of a really short duration.

In the last point when comparing both technologies we must evaluate the costs of each one. Although from the point of view of an engineer the costs associated to a specific device is not the most important aspect to consider it is sometimes an important fact that determines if a device is developed or not.

Although it is out of the scope of our diploma, another important fact could be to compare our protocol with other TDMA-based WAVE devices that our nowadays being implemented by other people. In Chapter 1 we mentioned some of them that were interesting for us when trying to define our protocol. The most of them based their analysis in distributed systems where there is not a central node or provider (which in our case in the RSU although it could also be an OBU) but any WAVE device has the ability to become a provider in case it wants to transmit a specific information to other WAVE devices. We could wonder which system, centralized or not, is better. The answer depends on the information we are interested in transmitting. From the point of view of the control channel if the safety information we want to send is, e.g., that the car in front of us has suddenly stopped, the safety message would be better handled in a distributed system because in this case the OBU that generates this emergency or critical message will be the one to act as a provider and send it to the other OBUs while in a centralised system the OBU should send it first to the RSU and this last one would be the one to communicate the message to rest of OBUs. But if the safety information is that the traffic light is going to turn red a centralised system will handle the information faster than a distributed one. The same problem occurs in the service channel. For example, in a weather forecast service a centralised system will have better results than in music sharing application service.

Appendix A
TCL script example

We consider it as important to include an example of a Tcl script which uses our protocol in the Appendix. Although the parameters of the Shadowing model and Nakagami model are included, finally the simulation makes use of the Two Ray Ground model. We recommend paying attention to the values given to the parameters of the physical layer (Phy/WirelessPhy), and application channel. We decided to highlight the important parts with bold letters when trying to use our protocol. Anyway the script is a simple version of the one we used because only two OBUs are defined.

```
# ==========================================================
# PARAMETERS
# ==========================================================
Phy/WirelessPhy set CSThresh_       3.162e-14
Phy/WirelessPhy set Pt_             0.001
Phy/WirelessPhy set freq_           5.9e9
Phy/WirelessPhy set L_              1
Phy/WirelessPhy set RXThresh_       3.652e-14
Phy/WirelessPhy set bandwidth_      10e6
Phy/WirelessPhy set CPThresh_       10.0
Phy/WirelessPhy set noise_floor_    7.96159e-14
#==========================================================
#configure RF model parameters
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

#Shadowing propagation model
Propagation/Shadowing set pathlossExp_ 2.7
Propagation/Shadowing set std_db_ 5.0
Propagation/Shadowing set dist0_ 1.0
Propagation/Shadowing set seed_ 0

#Nakagami propagation model
Propagation/Nakagami set use_nakagami_dist_ false
Propagation/Nakagami set gamma0_ 1.9
Propagation/Nakagami set gamma1_ 3.8
Propagation/Nakagami set gamma2_ 3.8

Propagation/Nakagami set d0_gamma_ 200
Propagation/Nakagami set d1_gamma_ 500
```

```
Propagation/Nakagami set m0_  1.5
Propagation/Nakagami set m1_  0.75
Propagation/Nakagami set m2_  0.75

Propagation/Nakagami set d0_m_ 80
Propagation/Nakagami set d1_m_ 200
#===========================================================
# Define options
#===========================================================
set val(chan)        Channel/WirelessChannel          ;# channel type
set val(prop)        Propagation/TwoRayGround         ;# radio-propagation model
set val(netif)       Phy/WirelessPhy                  ;# network interface type
set val(mac)         Mac/Tdma2                        ;# MAC type
set val(ifq)         Queue/DropTail/PriQueue          ;# interface queue type
set val(ll)           LL                              ;# link layer type
set val(ant)          Antenna/OmniAntenna             ;# antenna model
set val(ifqlen)      50                               ;# max packet in ifq
set val(nn)            2                              ;# number of mobilenodes
set val(roadunits)    1                               ;# number of roadside units
set val(rp)           DumbAgent                       ;# routing protocol
set opt(sc)   "/home/cris/workspace/ns2/5n.txt"       ;# node movement file.
set opt(x)      7500                                  ;# x coordinate of topology
set opt(y)      20                                    ;# y coordinate of topology
set opt(seed)   0.5                                   ;# seed for random number gen.
set opt(stop)   60.0                                  ;# time to stop simulation
# ==========================================================
# Main Program
# ==========================================================
# check for boundary parameters and random seed

if { $opt(x) == 0 || $opt(y) == 0 } {
        puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}

# Initialize Global Variables
set ns_          [new Simulator]
set tracefd  [open /home/cris/workspace/ns2/nodos.tr w]
set namtrace [open /home/cris/workspace/ns2/nodos.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

#Define a 'finish' procedure
proc finish {} {
      global ns_ tracefd
      $ns_ flush-trace
```

```
    #Close the trace file
    close $tracefd
    #Execute nam on the trace file
    #exec nam nodos.nam &
    exit 0
}
# set up topography object
set topo      [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# Create God
set god_ [create-god [expr $val(nn)+$val(roadunits)]]

# configure node
    $ns_ node-config -adhocRouting $val(rp) \
                     -llType $val(ll) \
                     -macType $val(mac) \
                     -ifqType $val(ifq) \
                     -ifqLen $val(ifqlen) \
                     -antType $val(ant) \
                     -propType $val(prop) \
                     -phyType $val(netif) \
                     -channelType $val(chan) \
                     -topoInstance $topo \
                     -agentTrace ON \
                     -routerTrace ON \
                     -macTrace ON \
                     -phyTrace ON \
                     -movementTrace ON
      set ID_(0) 0
      set rsu_(0) [$ns_ node]
      $rsu_(0) random-motion 0            ;# disable random motion
      $rsu_(0) set id_  $ID_(0)           ;# for unicast communication
      $rsu_(0) set address_ $ID_(0)
      $rsu_(0) nodeid $ID_(0)

      set ID_(1) 1
      set node_(1) [$ns_ node]
      $node_(1) random-motion 0           ;# disable random motion
      $node_(1) set id_  $ID_(1)
      $node_(1) set address_ $ID_(1)
      $node_(1) nodeid $ID_(1)

      set ID_(2) 2
      set node_(2) [$ns_ node]
      $node_(2) random-motion 0           ;# disable random motion
      $node_(2) set id_  $ID_(2)
      $node_(2) set address_ $ID_(2)
      $node_(2) nodeid $ID_(2)
```

```
# Provide initial co-ordinates and movements for mobilenodes
puts "Loading scenario file..."
source $opt(sc)

   # Define the node size in nam, adjust it according to our scenario
   $ns_ initial_node_pos $rsu_(0) 2
   $ns_ initial_node_pos $node_(1) 4
   $ns_ initial_node_pos $node_(2) 4

   set agent_(0) [new Agent/PBC3]
   $ns_ attach-agent $rsu_(0)  $agent_(0)
   $agent_(0) set periodicBroadcastInterval 0.003
   $agent_(0) set periodicBroadcastVariance 0.000
   $agent_(0) set payloadSize 1000
   $agent_(0) set nodeIdentifier 0
   $agent_(0) set servicesNumber 1
   $agent_(0) set numberUnicastServices 0
   $agent_(0) set numberFrames 5
   $agent_(0) PeriodicBroadcast ON
   $ns_ at $opt(stop).0 "$rsu_(0) reset";

   set agent_(1) [new Agent/PBC3Sink]
   $ns_ attach-agent $node_(1)  $agent_(1)
   $agent_(1) set periodicBroadcastInterval 0.00005
   $agent_(1) set periodicBroadcastVariance 0.006
   $agent_(1) set payloadSize 1000
   $agent_(1) set nodeIdentifier 1
   $agent_(1) set serviceReceived 1
   $agent_(1) PeriodicBroadcast ON
   $ns_ at $opt(stop).0 "$node_(1) reset";

   set agent_(2) [new Agent/PBC3Sink]
   $ns_ attach-agent $node_(2)  $agent_(2)
   $agent_(2) set periodicBroadcastInterval 0.00005
   $agent_(2) set periodicBroadcastVariance 0.006
   $agent_(2) set payloadSize 1000
   $agent_(2) set nodeIdentifier 2
   $agent_(2) set serviceReceived 1
   $agent_(2) PeriodicBroadcast ON
   $ns_ at $opt(stop).0 "$node_(2) reset";


$ns_ at $opt(stop).1 "finish"
$ns_ at $opt(stop).0 "puts \"NS EXITING...\" ; $ns_ halt"
puts "Starting Simulation..."
$ns_ run
```

Appendix B
Trace file obtained from Tcl Script of Appendix A

Once we execute the Tcl script of Appendix A we obtain a trace file (called nodos.tr). Here we show a fragment of these trace file. The first part corresponds to the control channel interval, we can see how the provider (identifier _0_) sends broadcast frames to OBUs one (id=_1_) and two (id=_2_). The second part corresponds to the service channel interval, it is possible to see how both OBUs send a request frame (generated by application protocol 'PBC3Sink') to the provider and how the last one send the information to the OBU whose request frame arrived first. For example, OBU 1 send the request frame number 6275 and OBU 2 send the request frame number 6276, both are received correctly by the provider but, in this case, the OBU 2 will be the one to receive the data (6277, 6278 and the next ones). To see in detail the meaning of each trace field read carefully the point 9.3 called "trace format" of [29].

```
s 18.705245397 _0_ AGT  --- 6259 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
r 18.705245397 _0_ RTR  --- 6259 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
s 18.705245397 _0_ RTR  --- 6259 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
s 18.705600000 _0_ MAC  --- 6259 PBC3 1052 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
s 18.708245397 _0_ AGT  --- 6260 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
r 18.708245397 _0_ RTR  --- 6260 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
s 18.708245397 _0_ RTR  --- 6260 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
r 18.708448749 _2_ MAC  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.708449067 _1_ MAC  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.708473749 _2_ RTR  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.708473749 _2_ AGT  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.708474067 _1_ RTR  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.708474067 _1_ AGT  --- 6259 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
s 18.708600000 _0_ MAC  --- 6260 PBC3 1052 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
s 18.711245397 _0_ AGT  --- 6261 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
r 18.711245397 _0_ RTR  --- 6261 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
s 18.711245397 _0_ RTR  --- 6261 PBC3 1000 [0 0 0 0] ------- [0:0 -1:0 32 0]
r 18.711448749 _2_ MAC  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.711449067 _1_ MAC  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.711473749 _2_ RTR  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.711473749 _2_ AGT  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.711474067 _1_ RTR  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
r 18.711474067 _1_ AGT  --- 6260 PBC3 1000 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
s 18.711600000 _0_ MAC  --- 6261 PBC3 1052 [0 ffffffff 0 800] ------- [0:0 -1:0 32 0]
```

```
s 18.750314050 _2_ AGT  --- 6275 PBC3Sink 20 [0 0 0 0] ------- [2:0 0:0 32 0]
r 18.750314050 _2_ RTR  --- 6275 PBC3Sink 20 [0 0 0 0] ------- [2:0 0:0 32 0]
s 18.750314050 _2_ RTR  --- 6275 PBC3Sink 20 [0 0 0 0] ------- [2:0 0:0 32 0]
s 18.750600000 _2_ MAC  --- 6275 PBC3Sink 72 [5 0 2 800] ------- [2:0 0:0 32 0]
D 18.750600318 _1_ MAC  --- 6275 PBC3Sink 72 [5 0 2 800] ------- [2:0 0:0 32 0]
r 18.750832742 _0_ MAC  --- 6275 PBC3Sink 20 [5 0 2 800] ------- [2:0 0:0 32 0]
r 18.750857742 _0_ AGT  --- 6275 PBC3Sink 20 [5 0 2 800] ------- [2:0 0:0 32 0]
s 18.752454019 _1_ AGT  --- 6276 PBC3Sink 20 [0 0 0 0] ------- [1:0 0:0 32 0]
r 18.752454019 _1_ RTR  --- 6276 PBC3Sink 20 [0 0 0 0] ------- [1:0 0:0 32 0]
s 18.752454019 _1_ RTR  --- 6276 PBC3Sink 20 [0 0 0 0] ------- [1:0 0:0 32 0]
s 18.753000000 _1_ MAC  --- 6276 PBC3Sink 72 [5 0 1 800] ------- [1:0 0:0 32 0]
D 18.753000318 _2_ MAC  --- 6276 PBC3Sink 72 [5 0 1 800] ------- [1:0 0:0 32 0]
r 18.753233059 _0_ MAC  --- 6276 PBC3Sink 20 [5 0 1 800] ------- [1:0 0:0 32 0]
s 18.753245397 _0_ AGT  --- 6277 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.753245397 _0_ RTR  --- 6277 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
s 18.753245397 _0_ RTR  --- 6277 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.753258059 _0_ AGT  --- 6276 PBC3Sink 20 [5 0 1 800] ------- [1:0 0:0 32 0]
s 18.753600000 _0_ MAC  --- 6277 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
D 18.753601059 _1_ MAC  --- 6277 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
s 18.756245397 _0_ AGT  --- 6278 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.756245397 _0_ RTR  --- 6278 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
s 18.756245397 _0_ RTR  --- 6278 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.756464742 _2_ MAC  --- 6277 PBC3 1000 [5 2 0 800] ------- [0:0 2:0 32 2]
r 18.756489742 _2_ AGT  --- 6277 PBC3 1000 [5 2 0 800] ------- [0:0 2:0 32 2]
s 18.756600000 _0_ MAC  --- 6278 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
D 18.756601059 _1_ MAC  --- 6278 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
s 18.759245397 _0_ AGT  --- 6279 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.759245397 _0_ RTR  --- 6279 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
s 18.759245397 _0_ RTR  --- 6279 PBC3 1000 [0 0 0 0] ------- [0:0 2:0 32 2]
r 18.759464741 _2_ MAC  --- 6278 PBC3 1000 [5 2 0 800] ------- [0:0 2:0 32 2]
r 18.759489741 _2_ AGT  --- 6278 PBC3 1000 [5 2 0 800] ------- [0:0 2:0 32 2]
s 18.759600000 _0_ MAC  --- 6279 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
D 18.759601058 _1_ MAC  --- 6279 PBC3 1058 [5 2 0 800] ------- [0:0 2:0 32 2]
```

Appendix C
PERL script used to read trace files

To obtain easily results from the trace files we opted to define a PERL that reads the traces to get information and prints this information in a text file. Here we show a simple example used to obtain the latency for Measure 1 (defined in Chapter 6) when the provider offers one unicast service and there are two OBUs who want it.

```
#!/usr/bin/perl

for($count=1; $count<101; $count++){          # TCL is run 100 times

system("./ns prueba4.tcl >> out.txt");        #Open the trace file
open(TRACE,"./../nodos.tr")||die "Could't open trace file: $!";

$numbersentnode1=0;
$numbersentnode2=0;

$numberconsumed1=0;
$numberconsumed2=0;

while((<TRACE>)){                                 # Get a line
        chomp;                                    # Clean it
        @words=split;                             # Chop it up

if (($words[0] eq "s") && ($words[2] eq "_0_") && ($words[3] eq "MAC")
     && ($words[9] eq "1")){           #Provider sends Unicast frame
                                         to OBU 1

            if($numbersentnode1<4){
                  if ($numbersentnode1==0) $txtimenode1=$words[1];

                  $numbersentnode1++;
            }
            else
            $numbersentnode1=0;

      }
elsif (($words[0] eq "s") && ($words[2] eq "_0_") && ($words[3] eq
        "MAC") && ($words[9] eq "2")){ #Provider sends Unicast frame
                                        to OBU 2

            if($numbersentnode2<4){
                  if ($numbersentnode2==0) $txtimenode2=$words[1];

                  $numbersentnode2++;
            }
            else
            $numbersentnode2=0;
      }
elsif (($words[0] eq "r") && ($words[2] eq "_1_") && ($words[3] eq
        "MAC") && ($words[9] eq "1")){  #OBU 1 receives the last frame


            if ($numberconsumed1==4){   # we begin counting in 0
            $rxtimenode1=$words[1];
            $requiredtimenode1=$rxtimenode1-$txtimenode1;
            $numberconsumed1=0;
            open(RES,">>1slot_2obu.txt")||die "Could't append: $!";
            print RES "$txtimenode1 $requiredtimenode1 $words[9]\n";
            }
            else{
            $numberconsumed1++;
            }
      }
elsif (($words[0] eq "r") && ($words[2] eq "_2_") && ($words[3] eq
        "MAC") && ($words[9] eq "2")){  #OBU 2 receives the last frame


            if ($numberconsumed2==4){    # we begin counting in 0
            $rxtimenode2=$words[1];
            $requiredtimenode2=$rxtimenode2-$txtimenode2;
            $numberconsumed2=0;
            open(RES,">>1slot_2obu.txt")||die "Could't append: $!";
            print RES "$txtimenode2 $requiredtimenode2 $words[9]\n";
            }
            else{
            $numberconsumed2++;
            }
      }
}
close(TRACE); # Close the trace
close(RES);
system("rm out.txt");
}
```

Appendix D
PERL script used to generate graphics

Once we have read the trace files with a PERL script, we will use another PERL script to use the information generated by the first script to get some graphics. If we pay attention to the bold lines we will see how firstly we open the text file obtained from the PERL script of Appendix C to get the information and work with it. Once we have made some calculations, in this case related to obtaining the latency from OBU 1, we write these values in another text file. The values of this second text file will be the coordinates (x-y pairs of values) that will be used to generate the graphic. Finally a graphic is obtained by using the GD::Graph tool [47].

```
#!/usr/bin/perl -w

use CGI ':standard';
use GD::Graph::bars;
use GD::Graph::colour;

$sumvalues{'1 OBU'}=0;
$sumvalues{'2 OBU'}=0;

$timesuser{'1 OBU'}=0;
$timesuser{'2 OBU'}=0;

open(RESULTS, "1slot_2obu.txt")||die "Could't open results file: $!";

      while((<RESULTS>)){                               # Get a line
      chomp;                                            # Clean it
      @words=split;                                     # Chop it up

      if($words[2]==1){
            $sumvalues{'1 OBU'}=$words[1]+$sumvalues{'1 OBU'};
            $timesuser{'1 OBU'}++;
      }

      }
      close(RESULTS);


$averageuser{'1 OBU'}=((($sumvalues{'1 OBU'}/$timesuser{'1 OBU'})*1000));

open(SUM,">summary_latency.txt")||die "Couldn't open summary: $!";

@keys=sort(keys(%averageuser));

foreach $key (@keys){
      push(@values,$averageuser{$key});
      print SUM "$key $averageuser{$key}.\n";
}
@data = ([@keys],[@values]);

close(SUM);

$mygraph = GD::Graph::bars->new(500, 300);
$mygraph->set(
    x_label     => 'Number of OBUs that receive unicast service',
    y_label     => 'Average time (msec)',
    title       => 'Multiplexation of one unicast service between one
                    or more OBUs',
    y_max_value => 20,
    y_min_value => 14,
    bgclr => 'black',
    shadowclr => 'grey',
    dclrs => [qw(lblue green)],
    bar_width => 50,


) or warn $mygraph->error;

$myimage = $mygraph->plot(\@data) || die $mygraph->error;

#write graph to a file
$bar_file = "latency.gif";
open(IMG, ">$bar_file")
  || die ("\nFailed to save graph to file: $bar_file. $!");

binmode(IMG);
print IMG $myimage->gif();
close(IMG);
```

Appendix E
Theoretical maximum latency calculation

We are going to explain in detail how the theoretical maximum latency is obtained in formula (6.5) of Chapter 6. The formula (6.5) defines the maximum latency in case of analysis A, that is when the RSU offers one service during the service channel interval. In this case there are 44 ms available for the RSU to send service data frames to the OBUs, see page 62. The formula (6.5) is used to obtain the maximum latency (or the worst latency) for OBU 5 in case all the OBUs (five in total) are interested in the service. This maximum latency will take place in case OBU 5 is the last one to send the acknowledge frame and hence it is the last one to consume the service.

As it is explained in Chapter 3 the RSU needs 2.712 ms to send a frame of 1000 bytes and the payload of any service is of 5000 bytes, therefore it is necessary five frames of 1000 bytes to send the whole data of a specific service. In this case any OBU will require 5x 2.712 ms ≈ 14 ms to consume the service, this means that in a service channel time slot there is enough time for three OBUs to consume the service:

$$14 \text{ ms x } 3 \text{ OBUs} = 42 \text{ ms} < 44 \text{ ms}$$

Considering all this information we can draw a simple scheme to obtain the maximum latency for OBU 5:

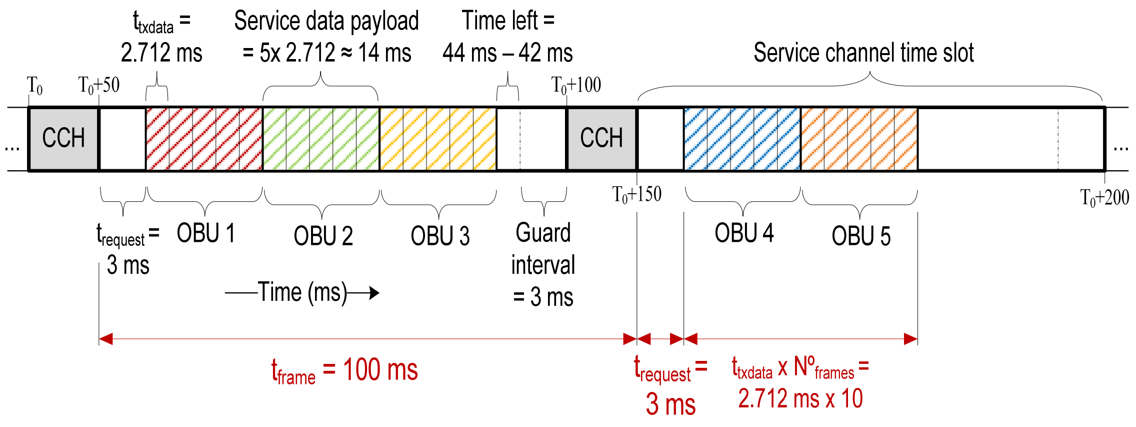


Figure Appendix E.1: Calculation of the maximum latency without time losses

As we can see in the figure the time necessary to consume the service for OBU 5 when Measure 3 is used will be:

$$latency = t_{frame} + t_{request} + (t_{txData} \times N^{o}_{frames}) = 100ms + 3ms + (2.712ms \times 10) \approx 131ms$$

This corresponds to the formula (6.5) used. If we consider in the analysis the time losses (of 3 ms, see Chapter 6) introduced by the RSU once the request interval finishes and when each OBU finishes consuming the service, we will have:

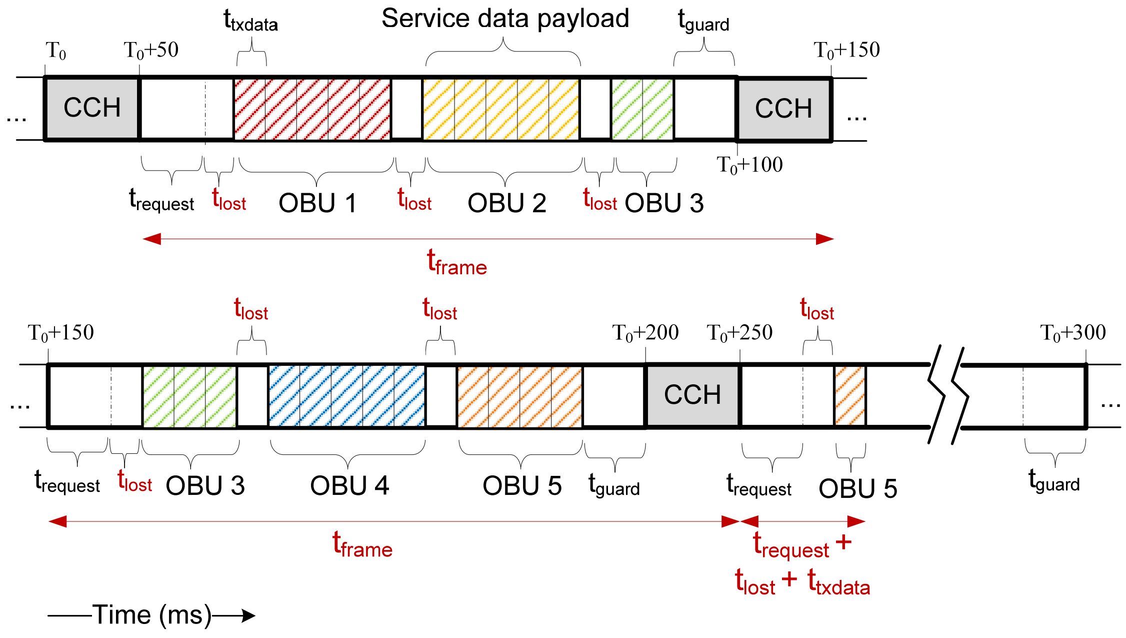


Figure Appendix E.2: Calculation of the maximum latency including time losses

In this case the total latency corresponds to the formula (6.6):

$$latencyWithLosses = 2t_{frame} + t_{request} + t_{losses} + t_{txData} = 200ms + 3ms + 3ms + 2.712ms \approx 209ms$$

Appendix F
Definition of the propagation model

As it is explained in Chapter 4 it is important to define the propagation channel because it strongly influences the communication window of each OBU. Among all the propagation models included in the actual version of the NS used (ns 2.33) we decided to make use of the Two-Ray Ground Reflection model. We are going to explain in detail how this propagation model is defined.

Basically the aim of a propagation model is to define the characteristics of the medium where the signal is going to be transmitted. These characteristics will influence the received signal power:

$$P_r(dBm) = P_t(dBm) - L(dB) \rightarrow p_r = \frac{p_t}{l} \quad (1)$$

P_r is the signal power in the receiver side, P_t is the signal power in the transmitter side and L are the losses introduced by the medium. The main difference among propagation models is how the losses introduced by the medium are defined.

The simplest propagation model is the Free Space model where the characteristics of the medium used to transmit the signal are not considered. This model is nowadays used to obtain the minimum propagation loss expected in a communication channel. If we consider isotropic antennas the basic losses are:

$$l_{basic} = \frac{p_t}{p_r} = \left(\frac{4\pi d}{\lambda}\right)^2 \quad (2)$$

In case the antennas are not isotropic we must include their gains in Formula (2):

$$l_{total} = \left(\frac{4\pi d}{\lambda}\right)^2 \cdot \frac{1}{g_t g_r} \quad (3)$$

And therefore the received signal power is:

$$p_r = p_t \cdot \left(\frac{\lambda}{4\pi d}\right)^2 \cdot g_t \cdot g_r \quad (4)$$

Formula (4) is called Friis equation. When the signal is not transmitted in Free Space conditions the medium will affect the signal by introducing attenuation. This attenuation is due to different causes like the existence of obstacles, fadings, influence of the rain and so on. The basic attenuation we must consider is the field attenuation which is defined by the formula:

$$a_e = \left(\frac{e_0}{e}\right)^2 \quad (5)$$

e_0 and e are the magnitude of the field produced by an isotropic antenna at a distance d when working under conditions of Free Space and in another type of medium respectively. If we include the field attenuation in Formula (3) we obtain:

$$l_{total} = \left(\frac{4\pi d}{\lambda}\right)^2 \cdot \frac{1}{g_t g_r} \cdot a_e \quad (6)$$

To obtain the field attenuation we have to define the propagation model we are going to work with. The Two-Ray Ground Reflection model considers the influence that the ground makes over the signal transmitted. When between the transmitter and the receiver there is a line of sight (which means they can see each other without visibility problems) and both of them are situated in the ground there will be two paths where the signal is transmitted: the direct path and the ground reflection path. For frequencies lower than 150 MHz there will also be surface waves, these waves will be predominant in case the frequency is lower than 10 MHz or the signal polarization is vertical.

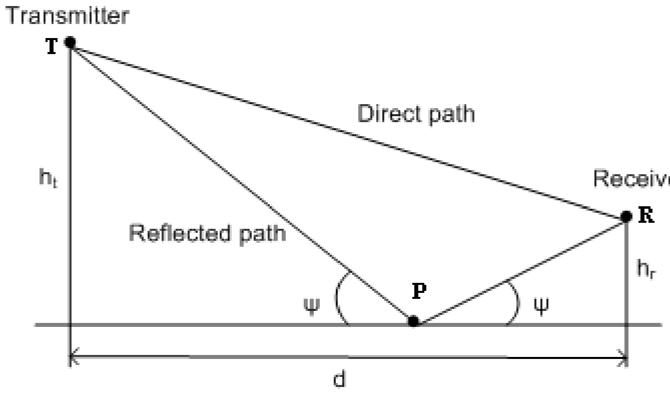


Figure Appendix F.1: Two-Ray Ground reflection model for flat earth conditions

In general the magnitude of the field in the receiver side when considering the Two-Ray Grougn reflection model is:

$$e = e_0\left[1 + R \cdot \exp(-j\Delta) + (1 - R) \cdot A \cdot \exp(-j\Delta)\right] \quad (7)$$

The first term of Formula (7) corresponds to the direct path of the signal, the second term is the reflected path and the last one is the surface wave. R is the reflection coefficient of the ground, Δ is the phase difference between the direct and the reflected path and A is an attenuation coefficient associated to the surface waves. We must keep in mind R is a complex value:

$$R = |R|e^{-j\beta} \quad (8)$$

The phase difference (Δ) is obtained with Formula (9):

$$\Delta = \frac{2\pi \cdot \Delta l}{\lambda} \quad (9)$$

In our case Δl will be the difference between the direct and the reflected path:

$$\Delta l = TPR - TR = \left[d^2 + (h_t + h_r)^2\right]^{1/2} - \left[d^2 + (h_t - h_r)^2\right]^{1/2} \approx \frac{2h_t h_r}{d} \quad (10)$$

The approximation can be done when d > h_t , h_r . Using this value in Formula (9) we obtain:

$$\Delta = \frac{4\pi h_t h_r}{\lambda d} \quad (11)$$

When the distance between transmitter and receiver is not high we can assume we are working in flat earth conditions. As we are also working with frequencies higher than 150 MHz there will not exist surface waves, in this case the module of Formula (7) will be:

$$e = e_0\left|\left\{ 1 + |R| \cdot \exp[-j(\Delta + \beta)]\right\}\right| = e_0\left|\left\{ 1 + |R| \cdot \exp[-j(\gamma)]\right\}\right| e_0\left[1 + |R|^2 + 2|R|\cos(\Delta + \beta)\right]^{1/2} \quad (12)$$

If we define γ=Δ+β and we remember that the module of a complex number verifies $|z| = \sqrt{\mathrm{Re}^2(z) + \mathrm{Im}^2(z)}$ we have Formula (13):

$$e = e_0 \left| \{ \ 1 + |R| \cdot \exp[-j(\gamma)] \} \right| = e_0 \sqrt{(1 + |R|\cos(-\gamma))^2 + (j|R|\sin(-\gamma))^2} =$$
$$= e_0 \sqrt{(1 + |R|\cos(\gamma))^2 + (-j|R|\sin(\gamma))^2} = e_0 \sqrt{(1 + 2|R|\cos(\gamma) + |R|^2 \cos^2(\gamma)) + (|R|^2 \sin^2(\gamma))} =$$
$$= e_0 \sqrt{1 + |R|^2 + 2|R|\cos(\gamma)} = e_0 \left[1 + |R|^2 + 2|R|\cos(\Delta + \beta)\right]^{1/2} \quad (13)$$

Considering that in general the incident angle will be $\psi = tg^{-1}\left(\frac{h_t + h_r}{d}\right) \approx 0$ because h_t and h_r are smaller than the path distance d, we make the approximation $|R| \approx 1$ and $\beta \approx \pi$ obtaining that:

$$e = e_0 \left[1 + |1|^2 + 2 \cdot |1|\cos(\Delta + \pi)\right]^{1/2} = e_0 \sqrt{[2 + 2\cos(\Delta + \pi)]} = e_0 \sqrt{2[1 - \cos(\Delta)]} =$$
$$= e_0 \sqrt{4 \frac{[1 - \cos(\Delta)]}{2}} = 2e_0 \left|\sin(\frac{\Delta}{2})\right| = 2e_0 \left|\sin \frac{2\pi h_t h_r}{\lambda d}\right| \quad (14)$$

When $d \geq \frac{12 h_t h_r}{\lambda}$ we can make a lineal approximation of Formula (14):

$$e = e_0 \frac{4\pi h_t h_r}{\lambda d} \quad (15)$$

Therefore the attenuation of the field will be:

$$a_e = \left(\frac{e_0}{e}\right)^2 = \left(\frac{\lambda d}{4\pi h_t h_r}\right)^2 (16)$$

And the losses:

$$l_{total} = \left(\frac{4\pi d}{\lambda}\right)^2 \cdot \frac{1}{g_t g_r} \cdot a_e = \frac{d^4}{(h_t h_r)^2} \cdot \frac{1}{g_t g_r} \quad (17)$$

Finally the received signal power is:

$$p_r = p_t \cdot \frac{(h_t h_r)^2}{d^4} \cdot g_t \cdot g_r \quad (18)$$

In case $d < \frac{12 h_t h_r}{\lambda}$ the Two-Ray Ground reflection model is not used. In this case the received signal power is obtained by using Formula (4) which corresponds to the Friis formula for a Free Space conditions. If we want to include further losses (*l*) the receiver signal power will be:

$$If \quad d \geq \frac{12 h_t h_r}{\lambda} \rightarrow Two - Ray \ Ground \ Model: p_r = p_t \cdot \frac{(h_t h_r)^2}{d^4} \cdot \frac{g_t \cdot g_r}{l}$$

$$If \quad d < \frac{12 h_t h_r}{\lambda} \rightarrow Free \ Space \ Model: p_r = p_t \cdot \left(\frac{\lambda}{4\pi d}\right)^2 \cdot \frac{g_t \cdot g_r}{l}$$

All the explanation showed in Appendix F is based in the information given in [30] and [48]. If we want to make use of the Two-Ray Ground propagation model within the NS we should define the following parameters:

PARAMETER	VALUE
g_t	1
g_r	1
h_t	1.5 meters
h_r	1.5 meters
l	1

Table Appendix F.1: Values given to the parameters in the simulation tests.

The values used for the gains of both antennas (transmitter and receiver) are respect to isotropic antennas (they are not in decibels) and the system loss is also given in natural units. We must also clarify that the heights of the antennas shown in Table Appendix F.1 are respect to the node were they are situated. Therefore if we want to know the total height of each antenna (height respect to the ground) we must add the height of the node. In our case the height of each node is given in the node movement file (see [46] and Appendix A) althought it can also be defined directly in the TCL code (see Chapter IX of [31]). In our case we assume that the height of the node where the transmitter antenna (OBU) is situated is 4.5 meters and the height of the node where the receiver antenna (RSU) is situated is 0 meters. In general terms we could say that our test simulations are based in an OBU of 5.5 meters high and five RSUs of 1.5 meters high. The values of the transmitter signal power and the wavelength are given in Appendix A. Finally we must point out that our the values of Table Appendix F.1 correspond to the default values of the NS that can be found in the *ns-default.tcl* file [49].

Bibliography

[1] Yunpeng Zang, Erik Weiss, Lothar Stibor, Bernhard Walke, Hui Chen and Xi Cheng. *Opportunistic wireless internet access in vehicular environments using enhanced WAVE devices.* Chair of communication networks, RWTH Aachen university. International journal of hybrid information technology, vol. 1, no. 2, April 2008.

[2] Yunpeng Zang, Lothar Stibor, Bernhard Walke, Hans-Jürgen Reumerman and Andre Barroso. *A novel MAC protocol for throughput sensitive applications in vehicular environments.*

[3] IEEE trial-use standard for wireless access in vehicular environments (WAVE) – multi-channel operation. IEEE Std 1609.4-2006, pages 1-15, 2006.

[4] Long Le, Wenhui Zhang, Andreas Festag and Roberto Baldessari. *Analysis of approaches for channel allocation in Cat-to-Car communication*.

[5] IEEE trial-use standard for wireless access in vehicular environments (WAVE) – Networking services. IEEE Std 1609.3-2007.

[6] Dr. Massimiliano Lenardi. *C2C-CC Phy/Mac standardization requirements and considerations.* C2C-CC and ISO/CALM workshop, 24/10/2007.

[7] Fan Yu and Subir Biswas. *Self-configuring TDMA protocols for enhancing vehicle safety with DSRC based vehicle-to-vehicle communications*. IEEE journal on selected areas in communications, vol. 25, No, 8. October 2007.

[8] Katrin Bilstrup, Elisabeth Uhlemann, Erik G. Ström and Urban Bilstrup. *Medium access control schemes intended for vehicle communication.* Centre for Research on Embedded Systems, Halmstad University; Department of Signals

and Systems, Chalmers University of Technology, Volvo Technology Corporation. 10-12/Sept/2007.

[9] Fan Yu and Subir Biswas. *Impacts of radio access protocols on cooperative collision avoidance in urban traffic intersection scenarios.* Electrical and computer engineering department, Michigan state university. 1-4244-0264-6/2007 IEEE.

[10] Jimmi Grönkvist. *Distributed STDMA in Ad hoc network.* Swedish defence research agency.

[11] Injong Rhee, Ajit Warrier, Mahesh Aia and Jeongki Min. *Z-MAC: a hybric MAC for wireless sensor networks.* Department of computer science, North Carolina state university.

[12] Injong Rhee and Jeongki Min. *DRAND : distributed randomized TDMA scheduling for wireless ad-hoc networks.* Department of computer science, North Carolina state university.

[13] Wei Ye, John Heidemann and Deborah Estrin. *An energy-efficient MAC protocol for wireless sensor networks.*

[14] Gang Lu, Bhaskar Krishnamachari and Cauligi S. Raghavendra. *An adapative energy-efficient and low-latency MAC for data gathering in wireless sensor networks.* Department of electrical engineering, university of southern California. 0-7695-2132-0/2004 IEEE.

[15] Zhihui Chen, Ashfaq Khokhar. *Self organization and energy efficient TDMA MAC protocol by wake up for wireless sensor networks.* 0-7803-8796-1/2004 IEEE.

[16] Wang-Rong Chang, Hui-Tang Lin and Bo-Xuan Chen. *TrafficGather: an efficient and scalable data collection protocol for vehicular ad hoc networks.* 1-4244-1457-1/2008 IEEE.

[17] Christoph Schroth, Florian Dötzer, Timo Kosch, Benedikt Ostermaier and Markus Strassberger. *Simulating the traffic effects of vehicle-to-vehicle messaging systems.* BMW Group Research and Technology, Hanauerstrasse 46, 80992 Munich, Germany.

[18] Stephan Eichler, Benedikt Ostermaier, Christoph Schroth and Timo Kosch. *Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic.*

[19] SUMO simulator web page: http://sumo.sourceforge.net/

[20] Daniel Krajzewicz and Christian Rössel. *SUMO - Simulation of Urban MObility - User Documentation.*

[21] VISSIM simulator web page: http://www.fabermaunsell.com/MarketsAndServices/47/17/index.jsp

[22] GloMoSim simulator web page: http://pcl.cs.ucla.edu/projects/glomosim/

[23] NCTUns simulator web page: http://nsl10.csie.nctu.edu.tw/

[24] Shie-Yuan Wang and Chih-Che Lin. *NCTUns 5.0: A Network Simulator for IEEE 802.11(p) and 1609 Wireless Vehicular Network Researches.*

[25] OPNet++ web page: http://www.omnetpp.org/index.php

[26] TraNS simulator web page: http://trans.epfl.ch/

[27] M. Piórkowski M. Raya A. Lezama Lugo P. Papadimitratos M. Grossglauser J.-P. Hubaux. *TraNS: Realistic Joint Traffic and Network Simulator for VANETs.*

Laboratory for computer Communications and Applications (LCA), School of Computer and Communication Sciences EPFL, Switzerland.

[28] Network simulator web page: http://www.isi.edu/nsnam/ns/

[29] Eitan Altman and Tania Jiménez. *NS simulator for beginners.* Lecture notes, 2003-2004. Univ. de Los Andes, Mérida, Venezuela and ESSI, Sophia-Antipolis, France.

[30] *The NS manual.* Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. The VINT Project. December 21, 2008. URL: http://www.isi.edu/nsnam/ns/ns-documentation.

[31] Marc Greis. *Marc Greis' tutorial.* http://www.isi.edu/nsnam/ns/tutorial/nsintro.html

[32] Intelligent Transportation Systems Standards Fact Sheet. IEEE 1609- Family of Standards for Wireless Access in Vehicular Environments (WAVE). URL: http://www.standards.its.dot.gov/fact_sheetp.asp?f=80

[33] IEEE 802.11p/D4.02 Draft Standard for Information Technology, Telecommunications and Information exchange between systems, Local and metropolitan area networks, specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Amendment 7: Wireless Access in Vehicular Environments. September, 2008.

[34] *Summary of CSMA/CA technology.* http://sss-mag.com/pdf/802_11tut.pdf

[35] *Introduction to 802.11 MAC.* http://www.coe.montana.edu/ee/rwolff/EE543-05/Lectures%20fall05/class%201%20MAC%2080211.pdf

[36] IEEE 802.11: Wireless access in vehicular environments (WAVE). Standard for Information technology, Telecommunications and Information exchange between systems, Local and Metropolitan networks, specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Player (PHY) specifications, 2007.

[37] GMPLS simulation tools http://www.teleinformatyka.net.pl/nti/old/nti2006/papers/Korniak_Rozycki_NTI_2006.pdf

[38] NS2 Notebook: Multi-channel Multi-interface Simulation in NS2 (2.29) http://www.cse.msu.edu/~wangbo1/ns2/nshowto8.html

[39] Qi Chen, Felix Schmidt-Eisenlohr, Daniel Jiang, Marc Torrent-Moreno, Luca Delgrossi and Hannes Hartenstein. *Overhaul of IEEE 802.11 Modelling and Simulation in NS-2 (802.11Ext).* University of Karlsruhe (TH), Mercedes-Benz Research & Development North America. 02/01/2008.

[40] Information about Nakagami fading. http://en.wikipedia.org/wiki/Nakagami_fading

[41] Example of how to use Diffserv in NS2. http://www.eng.tau.ac.il/~miriama/NS2/ex/nslab_b_ds_proj_inst.ppt#490,34

[42] Definition of physical queues and virtual queues for Diffserv implementation in NS2. http://www.ietf.org/mail-archive/web/diffserv/current/msg01952.html

[43] *Implementing multiqueue to extend controlled load service in Ns2.* http://www.nabble.com/Implementing-multiqueue-to-extend-controlled-load-service-in%09Ns2-td18113632.html#a18113632

[44] *Design and verification of an IEEE 802.11e EDCF simulation Model in ns-2.26.* Sven Wiethölter and Christian Hoene. Technical University Berlin. Telecommunications Networks group. Berlin, November 2003.

[45] Evaluation of IEEE 802.11e WLANs.
http://www.cs.odu.edu/~mweigle/research/wireless/

[46] *WAVE-based communication in vehicle to infrastructure real-time safety-related traffic telematics.* Manuel Zaera, Forschungszentrum Telekommunikation Wien, August 2008.

[47] Information about how to use the GD: Graph tool to obtain graphics.
http://gdgraph.com/samples.html

[48] José María Hernando Rábanos. *Transmisión por radio.* Editorial universitaria Ramón Areces.

[49] *ns-default.tcl* file.
http://nsnam.cvs.sourceforge.net/nsnam/ns-2/tcl/lib/ns-default.tcl?view=markup