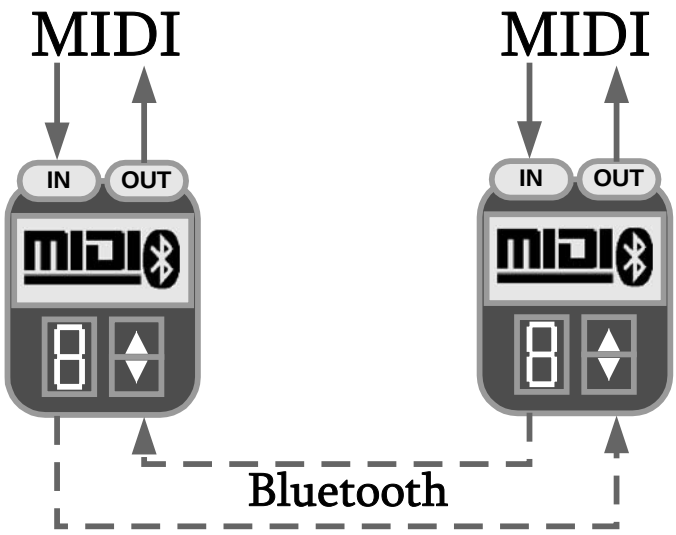
Universidade de Aveiro Departamento de Electrónica e Telecomunicações
2005

Paulo Jorge de Campos Bartolomeu

Avaliação de Bluetooth® para a transmissão sem fios de MIDI

Evaluating Bluetooth® for the wireless transmission of MIDI



Universidade de Aveiro Departamento de Electrónica e Telecomunicações
2005

Paulo Jorge de Campos Bartolomeu

Avaliação de Bluetooth® para a transmissão sem fios de MIDI

Evaluating Bluetooth® for the wireless transmission of MIDI

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. José Alberto Fonseca, Professor Associado do Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro.

dissertation submitted to the University of Aveiro in fulfilment of the thesis requirement for the degree of Master in Electronics and Telecommunications Engineering, under the supervision of José Alberto Fonseca, Associated Professor at the *Departamento de Engenharia Electrónica e Telecomunicações* of the University of Aveiro.

o júri

presidente

Prof. Doutor Alexandre Manuel Moutela Nunes da Mota
Professor Associado do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

Prof. Doutor Manuel Alberto Pereira Ricardo
Professor Auxiliar do Departamento de Engenharia Electrotécnica e Computadores da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor Paulo Maria Ferreira Rodrigues da Silva
Professor Auxiliar do Departamento de Comunicação e Arte da Universidade de Aveiro

Prof. Doutor José Alberto Gouveia Fonseca
Professor Associado do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

agradecimentos

Ao Professor Doutor José Alberto Gouveia Fonseca, queria agradecer a sua orientação exemplar e apoio incondicional.

Aos meus pais e irmão, pelos esforços que sempre empreenderam em meu auxílio nos diversos momentos difíceis da minha vida.

À minha namorada Patrícia, pela paciência e apoio.

palavras-chave

MIDI, indústria musical, espectáculos, redes sem fios *ad-hoc*, Bluetooth, SPP, ACL, pontualidade, Wi-FI, ZigBee, UWB

resumo

A indústria musical cresceu consideravelmente nos últimos anos originando uma forte procura de novas tecnologias que potenciem a flexibilidade dos espectáculos. Dado que o protocolo MIDI está fortemente implementado em instrumentos musicais, é necessário desenvolver soluções que permitam usufruir de flexibilidade na sua utilização. Esta dissertação procura endereçar esta problemática, recorrendo ao uso de tecnologias de transmissão sem fios. Em particular, é considerada a tecnologia *Bluetooth* para mapear as ligações MIDI tradicionais em ligações sem fios. Neste sentido, são propostas várias abordagens e destas são obtidos resultados indicando que a tecnologia *Bluetooth* pode ser uma boa opção no suporte de ligações MIDI sem recurso a fios.

keywords

MIDI, music industry, performances, *ad-hoc* wireless networks, Bluetooth, SPP, ACL, timeliness, WPANs, Wi-FI, ZigBee, UWB

abstract

During the past few years, the musical industry has grown considerably and has become more demanding in what concerns the use of technologies supporting flexibility in musical performances. Provided that MIDI is strongly implemented in the musical industry, it is required to develop solutions allowing flexibility usefulness. In the scope of this dissertation, an assessment of wireless technologies is performed. Particularly, *Bluetooth* is considered in the mapping of traditional wired MIDI connection in wireless MIDI links. In this sense, several approaches are proposed and results obtained, indicating that the *Bluetooth* technology may be suitable to support the wireless connection of MIDI devices.

Contents

Figures

Tables

CHAPTER 1

INTRODUCTION

1.1 Overview

Historically, musical performances date from 18.000 B.C. according to the earliest musical instruments found [1]. The oldest form of known written music dates from 3000 to 2000 years B.C. and was found on clay tablets retrieved from archaeological excavations in the Mesopotamian remains. Likewise, other civilizations such as the Egyptian or the Greek provide archaeological evidence of musical performances. Egyptian hieroglyphics comprise the earliest surviving representations of performing musicians while Greek iconography illustrates the emergence of other categories of musical performance such as dance or music for support of religious beliefs.

Musical performances have changed significantly since then, mainly due to cultural and technical evolution. In this sense, new types of performance have arisen to fulfill the urge for entertainment of modern audiences. Visual elements have been introduced to improve performances and, sometimes, to aid understanding the underlying message. Technology is

now playing a key role in some musical performances by enhancing sound, improving synchronization (for example synchronizing sophisticated stage pyrotechnics with the music itself), or just by allowing instruments to communicate with each other in a network. Some performances have grown into large networks of instruments, some producing notes or images and others synchronizing, enhancing, and mixing them to generate a set of synchronized sounds and visual effects. Despite of the underlying complexity and number of electronic instruments, a typical musical performance often occurs within a small geographic area, a stage.

The emergence of MIDI, as a *de facto* communication standard in the Musical Industry, has allowed solving some of the problems related with the use of synthesizers, sound modules or samplers: today it is possible to connect, via MIDI interfaces, many of these devices. This means that a single musician can control a range of very different sounds, making devices from a single master device (a keyboard, a guitar, etc.). In other words, it is possible to have a "full orchestra" in the stage being played by a handful of musicians. As pointed, Modern music performances also imply other aesthetic elements, such as image projection, movement, dance or theatre. This means that the musical stage is being transformed. It started as a static "storage space" for instruments and musicians and has evolved into a platform of multiple artistic expressions that needs to be flexible, clean, spacious and versatile.

This work has been developed envisioning a particular application in the Department of Communication and Art of the University of Aveiro. The target application consists of a theatrical music performance, Bach2Cage [2], where the mobility of musicians and instruments in the stage is extremely important. Bach2Cage is a set "tableaux" that occur in a stage under permanent change. Additionally, one of the musicians has reduced mobility and moves in the stage using a wheel chair. This led to the idea of using the keyboards on wheels as well, which led to particularly interesting ideas in choreographic and theatrical terms. The use of standard MIDI connections generates a large number of MIDI cables scattered around the stage, which creates a big problem in developing these ideas. A solution for some of these problems would be to use wireless technology for transmitting MIDI data. Besides the specific case of Bach2Cage, it was realized that wireless MIDI would be a very valuable resource for musicians and the music industry (in studios, touring concerts, music theatres, etc.).

This application field can be regarded as having real-time constraints in the sense that, if the communication fails, the performance degradation may cause the musical performance to collapse.

In the last few years several wireless technologies addressing low range communications have arisen. Examples of such technologies are Wi-Fi (802.11b, 802.11g), *Bluetooth* and ZigBee. These technologies share some common characteristics, such as the used spectrum, but have noticeable differences in terms of bandwidth, range and power consumption. However, there are available several commercial solutions for wireless MIDI Systems [3][4][5]. Nevertheless, they all share the same limitations: being expensive, power-hungry, large and proprietary.

This work explores *Bluetooth* as the MIDI enabling technology that will reduce the installation costs and inherent complexity. The transport of MIDI data streams using *Bluetooth* should also improve user experience and ergonomics given the well-known character of the technology.

1.2 Organization of the Dissertation

In this chapter we have outlined the motivation for this dissertation and briefly discussed the main objectives of the work. Moreover, several key technologies were identified as alternatives to the envisaged solution.

Because this work addresses the MIDI technology, chapter 2 presents a detailed analysis of the MIDI protocol. Besides describing the transmission medium and the protocol, this chapter also studies the MIDI file format and presents the most renowned software sequencers. The purpose of this study was to assess a platform that would allow the evaluation of the proposed system. The level of detail was found adequate since it could be decisive when considering options for implementation.

Chapter 3 discusses issues concerning the implementation of MIDI communications using wireless technology. An analysis of the timing requirements is here presented as well as its boundaries. This chapter also proposes the architecture of a Wireless MIDI system and discusses its features. Besides the analysis of the required timeliness, this chapter also identifies some of the available commercial solutions for the wireless transmission of MIDI.

Chapters 4 and 5 explore the technologies considered for supporting the wireless transmission of MIDI. Because *Bluetooth* was found more suited for the specified purpose, it is described in detail in chapter 4. The alternative technologies (Wi-Fi, ZigBee and UWB) are introduced in chapter 5. In the *Bluetooth* chapter, a vertical analysis is performed over all layers of the protocol stack while in chapter 5 the focus is on the lower layers. This differentiation occurs since, for the proposed architecture, three approaches are studied and each one uses different layers of the *Bluetooth* stack (ranging from the lower layers up to the higher layers). In addition, chapter 5 describes the main features of Wi-Fi,

ZigBee and UWB providing insight on their strengths and weaknesses concerning the application in analysis.

The proposed architecture is further described in chapter 6 and its operation is introduced. This chapter also describes the studied approaches to transport MIDI data flows (using *Bluetooth*) and the test-bed used to evaluate its performance. In this sense, results are presented and discussed in detail throughout each section addressing an individual approach.

Chapter 7 concludes this dissertation and provides some lines for future work and research.

CHAPTER 2

MIDI OVERVIEW

MIDI stands for Music Instrument Digital Interface and is a digital, non-proprietary hardware and software protocol for data communications among electronic musical instruments and computers. This protocol was firstly required in the early 80's [6] as a standard electronic "language" to enable musical keyboards from different manufacturers to control each other and to interface with computers and other hardware units.

In 1988 a standard MIDI file specification was defined, allowing files created by one MIDI software application to be used by other software applications. Later, in 1991, the General MIDI Standard version 1.0 (GM1) was developed by the MIDI Manufacturers Association (MMA) [7] and the Japanese Association of Musical Electronics Industry (AMEI) [8].

Since then, MIDI has experienced extreme popularity [9] mostly due to its enabling features as, for example, allowing MIDI files to be distributed and played with the original timbre choices on any general MIDI device.

However, GM1 was somehow constrained in what concerns the number of supported sounds (128). In this sense, Roland [10] developed an extension to the GM1 standard named GS (General Standard) supporting up to 16000 sounds. In 1998, following the increasing number of required sounds and features, MMA proposed the GM2 standard. In 2001, Roland and Yamaha agreed to participate and to support the GM2 standard [11]. GM2 includes several features of the Roland GS standard (support for reverb, chorus effects and larger number of reproducible sounds).

2.1 MIDI protocol

2.1.1 Introduction

MIDI is both a simple serial hardware interface and an elaborate protocol. The hardware interface operates at 31.25 Kbit/s ±1% using asynchronous serial communications with 1 start bit, 8 data bits and 1 stop bit per transmitted symbol (Figure 2-1).

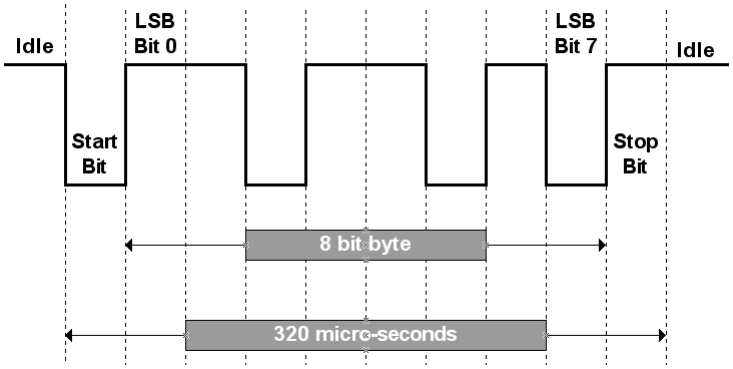


Figure 2-1 - MIDI timing for a single byte

The signaling and transmission bit order are similar to RS232, meaning that MIDI signaling is active-low and the Least Significant Bit (LSB) is the first to be sent on the channel. Symbols can represent MIDI commands or additional MIDI data according to their position in the MIDI stream.

2.1.1.1 MIDI Hardware and Software

Each MIDI instrument has several female 5-pin DIN jacks, which allow incoming or outgoing MIDI signals. These jacks are usually referred to as MIDI In (receives MIDI signals from another device), MIDI Out (transmits MIDI signals to another device) and

MIDI Through (captures MIDI signals on the MIDI In interface and replicates them on MIDI Through).

Male DIN MIDI cables are used to connect MIDI jacks from several MIDI instruments together. By connecting the MIDI Out of an instrument to the MIDI In of another instrument, a small MIDI network is made in which the keys pressed in the leftmost instrument are played in the rightmost instrument. This arrangement is shown in Figure 2-2.

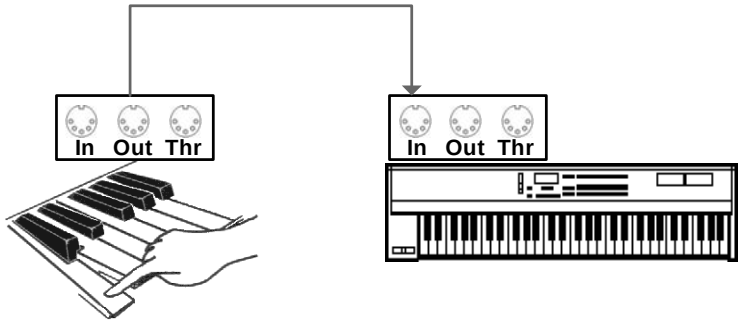


Figure 2-2 - Example of a MIDI setup for minimal operation

A particular feature of MIDI Through (usually identified as MIDI Thru) is that, when an instrument is being played, only the MIDI Out is driven. This means that, whenever an instrument creates or modifies MIDI signals, they are only sent to the MIDI Out interface. MIDI Thru serves the sole purpose of replicating the MIDI signal stream received at the MIDI In interface.

Obviously, MIDI interfaces can be used to build more complex instrument networks as shown in Figure 2-3.

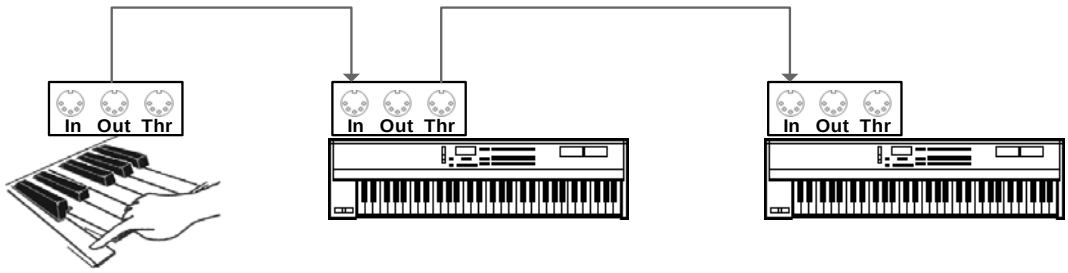


Figure 2-3 - Example of a MIDI setup for instrument daisy chaining

In this arrangement, keys pressed at the leftmost instrument will generate a data stream that will be sent to the remaining MIDI devices. The instrument at the center will receive the data stream and play it accordingly. Additionally, it will also replicate the received data

stream in its MIDI Thru interface allowing the rightmost instrument to play the same data stream.

MIDI can be used to record or playback musical performances whether by linking the MIDI Out of a musical instrument to the MIDI In interface of a sequencer (allowing to record the data stream of that particular instrument) or the MIDI Out of a sequencer to the MIDI In interface of an instrument (allowing that instrument to play the recorded data stream). Both arrangements are shown simultaneously in Figure 2-4.

MIDI sequencers allow recording musical performances by saving the actual MIDI data stream that the instruments are generating. They also allow the reverse functionality of playing a musical performance by sending over the MIDI Out interface the recorded data stream. One of the most interesting features of MIDI sequencers is that they usually have a *tempo* control, which allows them to play a musical performance slower of faster than the original recording. Unlike other digital audio playback systems this procedure does not change the pitch or the timbre of the sound.

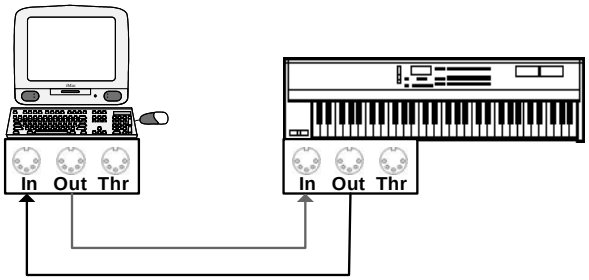


Figure 2-4 - Example of a MIDI setup for software sequencer operation

MIDI sequencers also provide other features like music editing on a per note basis, mute or solo individual parts, perform changes in timbre/volume, etc. Furthermore, sequencers have been improving their user interfaces allowing almost everyone to compose music in a What You See Is What You Get (WYSIWYG) fashion. This is a large improvement since a performer, whose musical technique is not sufficiently developed, may easily compose musical parts without the traditional Real-Time performance requirements.

Several sequencers are commercially available. However, when speaking about MIDI software sequencers, two manufacturers arise immediately: Cakewalk [12] and Steinberg [13]. Both have products directed to different market segments: home, intermediate and professional users. For example, Cakewalk provides *Home Studio* as an entry level software sequencer and, at a professional level, the more expensive *Sonar* (snapshot shown in Figure 2-5).

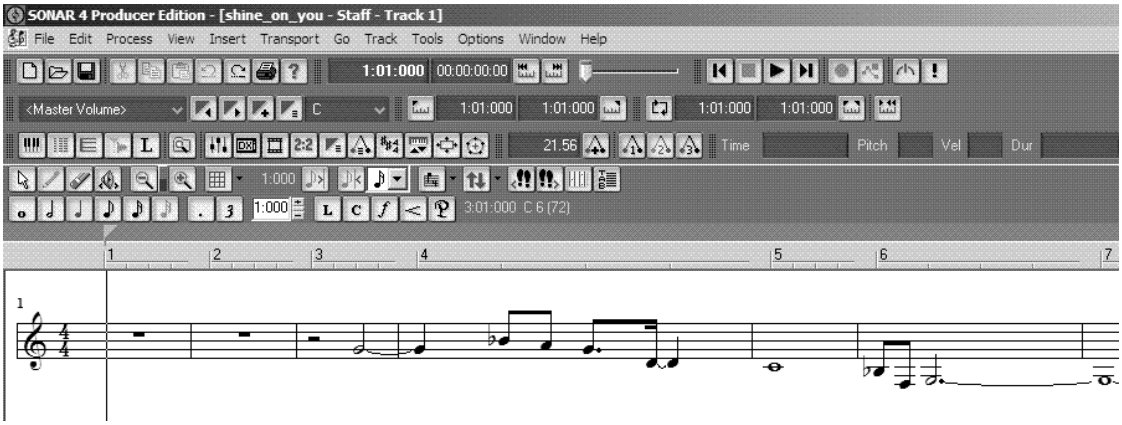


Figure 2-5 - Snapshot of the Cakewalk's Sonar 4 user interface

Today, most software sequencers store musical performances using a specific format, the Standard MIDI File format (SMF). This format will be discussed later in section 2.2 - Standard MIDI Files.

2.1.1.2 MIDI Commands

A MIDI command comprises a status byte followed by a variable number of data bytes (0-N). Status bytes are easily identified since their Most Significant Bit (MSB) is set. On the other hand, data bytes have their MSB unset and thus vary from 0 to 127. Status bytes identify mainly two types of messages: channel messages and system messages [14].

Channel messages use the upper nibble (4 MSBs) to encode the specific message and the lower nibble (4 LSBs) to address 1 of the 16 available MIDI channels. The base channel is a particular MIDI channel in which a device receives commands about the mode it should operate and the corresponding sound (voice). However, a device can receive messages in more than one channel depending on its operation mode.

There are two sub-types of channel messages: channel voice messages and channel mode messages.

- Channel voice messages are used to control individual voices;
- Channel mode messages are sent on the base address of the device and are used to specify its response to voice messages.

System messages are not (usually) targeted for specific MIDI channels and therefore are considered global messages. There are 3 basic groups of MIDI system messages: system exclusive (SysEx) messages, system common messages, and system real-time messages.

System exclusive messages are variable in length and have become the main route of expansion of the MIDI standard. These messages allow manufacturers to specify their own

messages for specific devices. Some authors such as [15] defend that *there is possibly more complexity in the various messages provided by this single status byte than by the rest of the MIDI specification*.

System common messages are primarily used for system setup operations. An example of these operations is the sequencer control and timing.

System real-time messages are always single byte commands that relate primarily to the control and timing of the sequencer playback. These messages always have priority over any other MIDI messages because they are used for sequencer synchronization and timing accuracy purposes.

Table 2-1 summarizes the most common MIDI status bytes. In channel voice (and mode) messages, the *nnnn* nibble can take values from 0 to 15 representing the 16 available MIDI channels. Likewise, as seen in Table 2-1, system common messages have status bytes with values ranging from 0xF1 to 0xF7 and system real-time messages with values ranging from 0xF8 to 0xFF.

Table 2-1 - Summary of Status Bytes

Status Byte		Number of Data Bytes	Description
Hex	Binary		
Channel Voice Messages			
8n	1000nnnn	2	Note OFF
9n	1001nnnn	2	Note ON (a velocity of zero = Note Off)
An	1010nnnn	2	Polyphonic Key Pressure (Aftertouch)
Bn	1011nnnn	2	Controller Change
Cn	1100nnnn	1	Program Change (instrument/voice selection)
Dn	1101nnnn	1	Channel Pressure (Aftertouch)
En	1110nnnn	2	Pitch Bend
System Messages			
F0	11110000	Variable	System Exclusive (SysEx)
F1-F7	11110sss	0 to 2	System Common
F8-FF	11111ttt	0	System Real-Time

The following subsection focuses “real” messages in the MIDI data stream by providing small examples for the channel voice messages illustrated in Table 2-1.

2.1.2 Channel Voice Messages

As stated before, channel messages are recognized by their first byte being in the 0x8n-0xEn range and having a variable number of data bytes, depending on the message type (1

or 2 bytes). These data bytes must have their MSBs set to 0 (otherwise they could be mistaken for status bytes).

2.1.2.1 Note ON and Note OFF

These are the most common messages in MIDI data streams when a performer is playing. The three bytes of the *Note On* message are identical to the MIDI *Note Off* (except for the high nibble) as seen in Table 2-2.

The *Note On* message instructs the instrument listening to channel *nnnn* to play the note *kkkkkkk* with a *vvvvvvv* velocity.

The MIDI standard defines the value 60 (decimal value) as the note "middle C". For each increment or decrement a note changes a semitone. Therefore, the decimal value 61 represents the C# note, value 62 the D note, and so forth. The wide note range that MIDI offers is adequate to support most musical instruments.

The velocity corresponds to the speed at which the key was pressed and its interpretation at the receiver varies according to the instrument. An example is the organ in which the velocity usually controls the loudness of the note (approximately constant). For other instruments (piano, for example) with more complex note behavior (attack, sustain and decay) the velocity can be interpreted as the maximum loudness of some particular note phase (attack phase).

Table 2-2 - Note Off and Note On

Note Off	
3 Bytes	1000nnnn, 0kkkkkkk, 0vvvvvvv
1000nnnn	Note Off Status byte; nnnn (0-15) = MIDI channels 1..16
kkkkkkk	Note number (0-127)
vvvvvvv	Key Off (release) velocity. Default value = 64.
Note On	
3 Bytes	1001nnnn, 0kkkkkkk, 0vvvvvvv
1001nnnn	Note On Status byte; nnnn (0-15) = MIDI channels 1..16
kkkkkkk	Note number (0-127)
vvvvvvv	Key On (attack) velocity. Default value = 64.

The *Note Off* message instructs the instrument listening channel *nnnn* to stop playing the note *kkkkkkk*. Clearly, there must be a perfect match between the *Note Off* and *Note On* messages in what concerns the playing note (the note must be the same in both MIDI messages). Otherwise, the instrument cannot stop the note that it is playing (it would try to stop another note that is not being played). The MIDI specification also requires a perfect

match between the velocity fields of the *Note Off* and *Note On* messages. However, most instruments accept a *Note Off* message with velocity 0 to stop playing a note.

2.1.2.2 Polyphonic Key Pressure

This message is seldom seen in MIDI data streams because it is only implemented in sophisticated keyboards. Usually, it is named as *aftertouch*, which is the pressure at the bottom of the key travel (for each key).

Table 2-3 - Polyphonic Key Pressure (aftertouch)

Polyphonic Key Pressure	
3 bytes	1010nnnn , 0kkkkkkk , 0vvvvvvv
1010nnnn	Polyphonic Key Pressure Status byte; nnnn (0-15) = MIDI channels 1..16
kkkkkkk	Note number (0-127)
vvvvvvv	Pressure value

Aftertouch information can be used to control modulation effects of separate notes (such as *vibrato*). The main reason behind the absence of this feature in regular keyboards is the use of additional pressure transducers. This clearly increases the overall cost of the keyboard (note that it would be needed at least a pressure sensor per key).

2.1.2.3 Controller Change

This message is used as a general-purpose channel message. In this sense, it carries control information about standardized “effects” such as breath, sustain, volume, etc. These are generated by the switches and controls of the instrument.

Table 2-4 - Control Change

Control Change	
3 Bytes	1011nnnn, 0ccccccc, 0vvvvvvv
1011nnnn	Control Change Status byte; nnnn (0-15) = MIDI channels 1..16
ccccccc	Controller number (0-119)
vvvvvvv	Controller value (0-127); For switch controllers : 0 = Off ,127 = On

Table 2-4 depicts the MIDI Controller Change message structure. The first data byte specifies the controller number and can be split in four groups:

- 000-063 High-resolution continuous controllers (0-31 = MSB; 32-63 = LSB);
- 064-069 Switches;
- 070-119 Low-resolution continuous controllers;
- 120-127 Channel Mode messages (see section 2.1.3).

The second data byte specifies the controller value. For switches, this byte is either 0 (Off) or 127 (On). For continuous, high/low resolution controllers this byte can take values from 0 to 127. As the name implies, these later controllers have a more accurate level of control in opposition to switches that only allow two levels: On or Off.

Low-resolution continuous controllers have a coarse adjustment while high-resolution controllers have an extended fine adjustment. A high-resolution controller has 16,384 adjustment levels (14 bits to encode the desired level) while low-resolution controllers have only 128 adjustment levels (7 bits to encode the desired level). Moreover, to use high-resolution controllers it is necessary to send two Controller Change messages, one for the coarse adjustment and the other for the fine adjustment.

As an example, consider the controller #1, the coarse adjustment for the Modulation Wheel and the controller #31, the fine adjustment for the same control. If this control is to be set on channel 0 to the level 8,197 (0x2005 = 0010 0000 0000 0101) two MIDI messages must be sent as depicted in Table 2-5.

There are a large number of defined controllers for this message (too large to mention here). Nevertheless, there are also controllers left for manufacturers to define specific instruments. As a final remark concerning controllers, there are two that worth an additional explanation: Registered Parameter Controllers (RPCs) and Non-Registered Parameter Controllers (NRPCs). These controllers use Registered Parameter Numbers (RPNs) and Non-Registered Parameter Numbers (NRPNs) to control specific internal parameters of a sequencer. RPNs differ from NRPNs in the fact that the former have to be registered with the MIDI Manufacturer's Association while NRPNs can be used by manufacturers according to their requirements. Clearly, because RPNs have to be registered they are part of the MIDI specification.

Table 2-5 - Setting the Modulation Wheel to the level 8,197

Coarse Adjustment	
10110000	Controller change on channel 0
00000001	Coarse adjustment of modulation wheel
00000100	value = bits 7 to 13 of 14-bit overall value.
Fine Adjustment	
10110000	Controller change on channel 0
00110011	Fine adjustment of modulation wheel
00000101	bits 0 to 6 of 14-bit overall value.

2.1.2.4 Program Change

The term “program” is generally interpreted as “patch” and refers to the current sound setup of the sequencer. Most sequencers support factory-defined or user-defined patches,

each one emulating a different instrument or timbre. The Program Change message enables switching the sequencer over these patches, thus producing different sounds when a MIDI Note On message is received.

Table 2-6 - Program Change

Program Change	
2 Bytes	1100nnnn, 0kkkkkkk
1100nnnn	Program Change Status byte; nnnn(0-15) = MIDI channels 1..16
kkkkkkk	Program number (0-127)

In most scenarios this is a MIDI message that will only be seen in the beginning of a music. However, when more than sixteen instruments are to be played in one music and there are only sixteen physical instruments or less, this message can be used to change the "patch" of some MIDI channel "on the fly" thus allowing the additional instruments to be played.

2.1.2.5 Channel Pressure (Aftertouch)

This message is quite similar to the Polyphonic Key Pressure message. The difference is that the Channel Pressure message provides average "aftertouch" information of all pressed keys in a specified instant (rather than on individual keys).

This average "aftertouch" information can be used to control the modulation effects of a sequencer.

Table 2-7 - Channel Pressure

Channel Pressure (Aftertouch)	
2 Bytes	1101nnnn, 0kkkkkkk
1101nnnn	Channel Pressure Status byte; nnnn(0-15) = MIDI channels 1..16
kkkkkkk	Channel Pressure (0-127)

2.1.2.6 Pitch Bend

Nowadays, a large number of MIDI controller keyboards have a "pitch wheel" allowing the player to "bend" notes as guitar players do. When the performer uses this wheel the MIDI controller generates a Pitch Bend message as shown in Table 2-8.

This message carries a 14-bit (least significant byte (LSB) plus most significant byte (MSB)) number that identifies the wheel position or, in other words, the "bend" that should be made to the current playing note. In the default position, the pitch wheel is "centered" and its value is $v_{\_center}=2^{14}/2=8192$ (0x2000). Notwithstanding, it can be used to reach

extreme values such as 0x0000 or 0x3FFF respectively. Clearly, the note “bend” effect is proportional to the deviation of the wheel from the center value.

Table 2-8 - Pitch Bend

Pitch Bend	
3 Bytes	1110nnnn, 0vvvvvvv, 0hhhhhhh
1110nnnn	Pitch Bend change Status byte; nnnn(0-15) = MIDI channels 1..16
vvvvvvv	Pitch Bend LSB value (0-127)
hhhhhhh	Pitch Bend MSB value (0-127)

2.1.2.7 Status Byte

Running Status

The running status is a special condition in which the MIDI message status byte is omitted for consecutive messages with the same status byte. This condition is used to save bandwidth when several (2 or more) consecutive messages have the same status byte. In this scenario, the status byte is only transmitted once and omitted in subsequent messages. The receiver associates the received bytes with the appropriate status byte. When a different status byte is received the running status condition is terminated.

Undefined or Unimplemented Status

The reception of undefined (or unimplemented) status bytes should be ignored. Furthermore, subsequent data bytes should also be ignored until the reception of the first valid status byte. This approach allows the future development of the MIDI specification without impact on the program of the device.

2.1.3 Channel Mode Messages

Channel Mode messages are a subset of Controller Change messages since they have the same status byte format (*0b1011nnnn*) but different controller range (120-127).

The MIDI specification defines four basic operation modes. These modes can be chosen using any two of the last four Channel Mode Messages (controllers 124-127) depicted in Table 2-9.

The Omni Mode defines if a device should respond to messages received on any channel (Omni Mode On) or only in its base channel (Omni Mode Off). The Mono mode defines whether a device is able to play just a single note at any time (Mono Mode On) or if it can play chords in a specified MIDI channel.

Table 2-9 - Channel Mode Messages

Channel Mode Messages		
3 Bytes		1011nnnn (nnnn=channel number), 0ccccccc, 0vvvvvvv
ccccccc	vvvvvvv	Description
120	0	All Sound OFF; all sound generators shall stop upon receiving this message. This operation doesn't involve any release phase
121	0	Reset All Controllers; this message resets all MIDI controllers to their default values
122	0(Off) / 127(On)	Local Control On/Off; the local control allows a device to control if the local data is sent to the MIDI OUT port and to the sound generator (On) or only to the MIDI OUT port (Off)
123	0	All Notes Off (ANO); causes sound generators to enter the release state, thus slowly stop making sound
124	0	OMNI Mode Off; this message causes ANO
125	0	OMNI Mode On; this message also causes ANO
126	0-16	Mono Mode On (Poly Off); also causes ANO; vvvvvvv indicates the number of channels to use (Omni Off). If this number is zero then it means that all available channels should be used (Omni On)
127	0	Poly Mode On (Mono Off); also causes ANO

Notice that devices solely “hear” MIDI Mode Messages on their base channels. This means that, even for the Omni Mode, a device will only respond to a MIDI Channel Mode message if it was received on its base channel.

The four basic operation modes are described as:

- ***Omni Mode On, Polyphonic Mode On***
  - Bn, 7D, 00 → Omni Mode On
  - Bn, 7F, 00 → Poly Mode On

  In this MIDI mode, messages are received from all voice channels and assigned to voices polyphonically. The transmission of polyphonic voice messages is performed using the base channel.
- ***Omni Mode On, Monophonic Mode On***
  - Bn, 7D, 00 → Omni Mode On
  - Bn, 7E, 00 → Mono Mode On

  In this MIDI mode, messages are received from all voice channels and assigned to a single voice (monophonically). The transmission of monophonic voice messages is performed using the base channel.

- ***Omni Mode Off, Polyphonic Mode On***
  - Bn, 7C, 00 → Omni Mode Off
  - Bn, 7F, 00 → Poly Mode On

In this MIDI mode, messages are received only in the base channel and assigned to voices polyphonically. The transmission of polyphonic voice messages is also performed using this channel.

- ***Omni Mode Off, Monophonic Mode On***
  - Bn, 7C, 00 → Omni Mode Off
  - Bn, 7E, nn → Mono Mode On (nn = number of channels to use)

  In this MIDI mode, messages are received on voice channels *base channel + nn - 1* and assigned to voices 1 to *nn* (monophonically). The transmission of monophonic voice messages (1 to *nn*) is performed using the base *channel + nn - 1* channels (one voice per channel).

2.1.4 System Messages

System messages are not associated with any particular channel. Instead, they are intended for the whole MIDI system. The next subsections describe, with more detail, the existing MIDI system message types.

2.1.4.1 System Exclusive Messages

System exclusive messages are used to transport information for/from a specific MIDI device. Equipment manufacturers generally describe particular sounds using these messages. The actual data sent in these messages is not usually usable by another device, e.g., the structure of the message can be very particular and not understandable by other devices (even from the same manufacturer).

System exclusive messages typically arise as result of a request issued by a device. However, MIDI devices are free to ignore SysEx messages (including its data bytes) that they do not understand or are not interested in. Table 2-10 shows the SysEx MIDI message structure.

When the identification code (2nd byte of the message) is zero the following two bytes are used as extensions to the manufacturer ID. The End Of System Exclusive (EOX) message will terminate a SysEx MIDI message. However, any other status byte (except System Real Time) will have the same effect. This feature prevents endless SysEx messages from happening when the EOX byte is not received.

Table 2-10 - System Exclusive Messages (SysEx)

System Exclusive Messages			
Status Byte		Data Bytes	Description
Hex	Binary		
F0	11110000		Start of System Exclusive (SOX)
		0iiiiiii	Identification code (0-127)
		0ddddddd … 0ddddddd	Any number of data bytes (each 0-127) having manufacturer specific functionality
F7	11110111		End of System Exclusive (EOX)

2.1.4.2 System Common Messages

System common messages, as the name suggests, are the most common system messages and are intended to all units of the MIDI system. Table 2-11 illustrates the structure of these messages.

Table 2-11 - System Common Messages

Channel Common Messages				
Status Byte		Description		
Hex	Binary			
F1	11110001	0nnndddd	---	MIDI Time Code Quarter Frame; nn=message type; dddd=value;
F2	11110010	0lllllll	0hhhhhhh	Song Position Pointer; lllllll=LSB (0-127); hhhhhhh=MSB (0-127); Number of MIDI beats (since the song start) is given by the 14-bit number
F3	11110011	0sssssss		Song Select; sssssss=Song Number (0-127)
F4	11110100			Undefined
F5	11110101			Undefined
F6	11110110	---	---	Tune Request; analogue synthesizers should start self tuning after receiving this message
F7	11110111	---	---	End of System Exclusive (EOX); this message is used to terminate a system exclusive message

2.1.4.3 System real-time messages

System real-time messages can be sent at any instant, even in the middle of other MIDI messages. When this occurs, the receiving device can choose to act in response to the received system real-time message or ignore it. However, after this procedure, the receiving device shall resume to the previous state. This mechanism supports an accurate timing maintenance of the MIDI system.

Table 2-12 - System Real-Time Messages

System Real-Time Messages		
Status Byte		Description
Hex	Binary	
F8	11111000	Timing Clock; if a MIDI device is playing this message is transmitted 24 times per quarter note
F9	11111001	Undefined
FA	11111010	Start; start playing the current sequence from the beginning
FB	11111011	Continue; continue playing the current sequence from the point it was stopped
FC	11111100	Stop; stop playing the current sequence
FD	11111101	Undefined
FE	11111110	Active Sensing; this message starts the active sensing state in the receiver. In this state the receiver expects to receive an active sensing message each 300ms (maximum). When the receiver does not receive it within the specified time window it assumes that the connection was terminated
FF	11111111	Reset; this message resets all receivers to their power-up status

Notice particularly the reset message in Table 2-12. Assuming that two devices were programmed to send this message on power-up, a deadlock scenario can occur since they can reset each other indefinitely. This message must be used with special care to avoid deadlock scenarios.

2.2 Standard MIDI Files

Standard MIDI files are a common file format used by several musical software and hardware devices to store information about songs. MIDI files contain the necessary information to reproduce a musical performance. In other words, they contain not only time information for each event, but also, information about the song such as the number of tracks, track structure, *tempo*, etc.

A standard MIDI File stores standard MIDI messages as well as the time-stamp for each one. The time-stamp information allows sequencers to play events according to the original timing. This information is represented by a series of bytes that store the number of clock pulses to wait before "playing" the event.

MIDI files were designed to be generic and compact. In this sense, they are usually small and supported by most sequencers. Because MIDI files contain information arranged in data chunks that can be loaded, parsed, skipped, etc. they allow a high degree of flexibility in what concerns the storage of proprietary data. This way, other sequencers can just parse the MIDI file and ignore unknown MIDI chunks.

2.2.1 File Structure

MIDI Files contain two basic chunks of information: header chunk and track chunk. Chunks are groups of bytes preceded by an ID and corresponding size. Each file contains one, and only one, header chunk and a variable number of track chunks.

A data chunk is always prefixed with an 8-byte chunk header. This chunk header contains a 4-byte ID string followed by a 4-byte field indicating the length of the chunk as the number of bytes that follow. The 4-byte ID string identifies the type, which can be MThd (MIDI File header) or MTrk (MIDI File track).

To better illustrate this; one can use the following chunk header example (as it would be seen in a hex editor):

4D 54 68 64 00 00 00 06

The initial 4 bytes identify the type, that is, the ASCII values for ‘M’, ‘T’, ‘h’ and ‘d’. The following 4 bytes indicate the remaining length of the chunk, meaning that, after this header, it should be expected to find 6 bytes before the end of the chunk. Notice that the length is in the “Big Endian“ byte order, e.g., Most Significant Bytes (MSBs) appear before Least Significant Bytes (LSBs).

2.2.1.1 MThd Chunk structure

The header chunk stores information regarding the format of the file, the number of tracks and the timing division. Each file contains only one MThd chunk and it always comes in first place (so that a sequencer or other application can easily identify its type).

A MIDI MThd chunk has always a 6-byte “payload” (as shown in Table 2-13) because the format type, the number of tracks, and the time division parameters occupy 2 bytes each.

The *Format Type* describes how the following track information shall be interpreted. A type 0 MIDI file has only one track containing all the events for the entire song. These

include the song title, time signature, *tempo*, etc. A type 1 MIDI file can have two or more tracks. The first track contains, by convention, all the song related information like the title, the time signature, etc. The second and following tracks contain track specific information such as the musical event data. A type 2 MIDI file contains multiple tracks. However, each track represents a different sequence, which may not be played simultaneously. Usually, this type of file is used to save songs with multi-pattern music sequences.

Table 2-13 - MIDI Header chunk format

MIDI Header Chunk Format			
Offset	Length	Description	Value
0x00	4	Chunk ID	"MThd" (0x4D546864)
0x04	4	Chunk Size	6 (0x00000006)
0x08	2	Format Type	0 - 2
0x0A	2	Number of Tracks	0 - 65535
0x0C	2	Time Division	Discussed in the following text

The *Number of Tracks* defines the number of track chunks that follow this header chunk. A type 0 MIDI file contains only one track, while type 1 and 2 files can contain up to 65535 tracks.

The *Time Division* parameter is used to decode the track event delta times into "real" time and represents either ticks per beat or frames per second depending on the top bit of the 2-byte word. Moreover, if the top bit is 0, the following 15 bits represent the time division in ticks per beat. Otherwise, the following bits describe the time division in frames per second. The number of frames per second is obtained by splitting these 15 bits into two parts. The 7 most significant bits represent the number of SMPTE[1] frames per second, which can be -24, -25, -29 or -30. These negative values are stored in two's complement form. The remaining 8 bits define how many clock ticks or track delta positions there are per frame. So a time division example of 0xE728 can be splitted in three parts. The most significant bit is 1 and therefore the time division is in the SMPTE frames per second format. The following 7 bits (0b1100111) represent the number of frames per second. Since the most significant bit of this set is 1 the represented number is negative, in this case -25 in decimal. Therefore there are 25 frames per second. Since the least significant byte is

[1] SMPTE Time Code (Society of Motion Picture and Television Engineers) was originally developed by NASA to synchronize computers together. SMPTE Time Code is a representation of absolute time in that it follows hours, minutes, seconds and frames just like a particular watch, thus allowing an exact timing reference.

0x28, we have 40 clock ticks (or track delta positions) per frame yielding a 1-millisecond tick resolution or, in MIDI files, a delta-time increment of 1 millisecond.

2.2.1.2 MTrk Chunk structure

A track chunk stores information about an individual track, including the track name and music events. Table 2-14 illustrates the format of the track chunk. A track chunk header starts with a chunk ID followed by the length of the chunk. After these fields, the individual track event data follows. It contains all MIDI data (including timing bytes and optional non-MIDI data - *tempo* settings, track names, etc.) for this individual track.

Table 2-14 - MIDI Track chunk format

MIDI Track Chunk Format			
Offset	Length	Description	Value
0x00	4	Chunk ID	"MTrk" (0x4D54726B)
0x04	4	Chunk Size	variable
0x08		track event data	

The chunk size is variable and depends on the number of bytes used for all events contained in the track. The track event data field contains MIDI events that characterize the sequence and how it is played.

Track chunks are made of events, each one preceded by its time-stamp. Each event time-stamp is referenced by the previous event. In other words, if an event occurred 6 clocks after another event, then its “delta-time” is 6. If the following event occurs simultaneously with the previous, then its “delta-time” is 0, and so forth. Therefore the delta-time is the duration (in clock ticks) between an event and its predecessor.

Delta-times are stored as a series of bytes called *variable-length quantity*. Each byte contributes with only its less significant 7 bits. Therefore, if we are using 32-bit delta-times we have to break it into a series of 7-bit bytes. The delta-time value dictates the number of bytes that are effectively used for storage. Obviously, larger delta-times will result in a larger number of storage bytes. To indicate which byte ends the series, its most significant bit will be left cleared while the remaining bytes have this bit set. For example if the largest delta-time allowed is in the range 0-127 then it can be represented by a single byte.

In an MTrk chunk, the first (1 to 4) bytes represent the first delta time of the event as a *variable-length quantity*. The following data is the actual first event. If this is a MIDI event, then it will be the actual MIDI Status byte.

2.2.1.3 MIDI events

A MIDI file is described using track events. Each event includes an individual set of parameters, namely the delta-time, event type and, normally, some kind of event specific data. Events can be grouped into three types: Control Events, System Exclusive Events and Meta Events. This section only discusses Meta Events since the remaining were previously addressed in former sections.

A particular characteristic of Meta Events is that they are not sent (or received) over a MIDI port. Therefore they are used primarily for optional features. Despite this fact, they are similar to other events in the sense that they also have a delta-time relative to the previous event and that they can be intermixed with other MIDI events.

Table 2-15 illustrates the structure of the Meta Event whose status byte has the 0xFF value. Notice that this value is also used in MIDI to specify a “reset” which, however, is never stored in a data file. After, the event type parameter defines the individual Meta Event and the length defines the actual number of parameter bytes that follow. Finally, we have the specific variable-length parameter data.

Table 2-15 - Structure of the Meta Events

Meta Events			
Meta Event (Status Byte)	Type	Length	Data
0xFF	0-255	Variable-length	Type specific

Table 2-16 presents a short overview of the commonly used Meta-Events and describes summarily each one.

The *Sequence Number* event specifies the MTrk sequence number and must be placed at the beginning of each MTrk. Nevertheless, it can assume two forms: specify *SS SS* or not. In the first case *SS SS* refers to the MIDI Cue message number. Afterwards, this number is (in a format 2 MIDI file) used so that a "song" sequence can refer to “patterns” (e.g., MTrk) using the MIDI Cue message. In the second case, when the *SS SS* number is omitted, the sequence number corresponds to the location of MTrk in the file. In a format 2 MIDI file only one of these events is allowed per MTrk chunk. In format 0 and 1 MIDI files, only one of these is allowed and it must be in the first MTrk.

All the *Text*, *Copyright*, *Track Name*, *Instrument Name*, *Lyric*, *Marker* and *Cue Point* events share the same structure. The third byte of these events specifies the number of symbols that follow and the remaining bytes are used to describe the event specific information (copyright notice, instrument name, etc.).

The *End of Track* event is not optional since every MTrk must end with one. This event is used as the definitive marking of an MTrk end.

As previously introduced, the MIDI file format specifies *tempo* as the amount of time (expressed in microseconds) per quarter note. A *Tempo* event indicates a *tempo* change. The parameter *TT TT TT* specifies the new *tempo* as the number of microseconds per quarter note. Of course, if we desired a *tempo* of 250 milliseconds, then *TT TT TT* should be equal to 0x03 0xD0 0x90. In this case each quarter note would be 250 milliseconds long.

Table 2-16 - Overview of the most common Meta Events

Status byte	Data byte		
	2nd byte	Other bytes	Description
FF	00	02 SS SS	Set the sequence of the track
FF	01	NN TT…	Any Text user wants
FF	02	NN TT…	Text for copyright info
FF	03	NN TT…	Track name
FF	04	NN TT…	Track instrument name
FF	05	NN TT…	Lyric
FF	06	NN TT…	Marker
FF	07	NN TT…	Cue Point
FF	2F	00	End of Track
FF	51	03 TT TT TT	Set *tempo* (microseconds/quarter note)
FF	54	05 HH MM SS FF SF hour/minute/second /frame/subframe	SMPTE Offset
FF	58	04 NN DD CC BB numerator/denominator/ metronome ticks/ 32nd notes per quarter note	Time signature
FF	59	02 SF MI key(sharp/flat) scale(0:major, 1:minor)	Key signature (C when Key=0)
FF	7F	7F	Sequencer Specific Information

The *SMPTE Offset* event specifies the start time of the MTrk in hours (*HH*), minutes (*MM*), seconds (*SS*), frames (*FF*) and subframes (*SF*). This event must be placed at the

beginning of the MTrk. The subframe field designates the number of fractional frames in 100ths of a frame.

The *Time Signature* event defines four parameters: numerator (*NN*), denominator (*DD*), metronome ticks and 32^{nd} notes per quarter note. The first two represent the signature as notated in the sheet music. The metronome ticks parameter represents the number of MIDI clocks in a metronome tick and the 32^{nd} notes per quarter note parameter represent the number of notated 32^{nd} notes in a quarter note.

The *Key Signature* event allows, as the name implies, the specification of the key signature.
Finally, the proprietary (*Sequencer Specific Information*) event can be used to store proprietary events. The first bytes should be chosen in a way that a program could easily evaluate if the event is addressed to it.

There are several Meta Events that can be used to improve the storage of musical performances. However, even if some desired functionality is not already defined, it can be implemented using the extension mechanisms provided in the SMF specification.

2.3 Conclusion

This chapter described MIDI by providing a deep overview of the protocol. It can be concluded that it is a very complex protocol to be implemented in an embedded manner. So, the approach of wirelessly transmit MIDI should be based on a solution handling MIDI commands transparently. This means that the approach to be considered should not lie on individual command interpretation.

The author believes that any feasible solution should not be supported on particular features of the protocol, since it would increase the complexity of the system and perhaps reduce its performance.

CHAPTER 3

ISSUES CONCERNING THE WIRELESS IMPLEMENTATION OF MIDI

Chapter 1 presented the motivation for this work by describing the envisaged application scenario and by identifying some of its requirements. Although existing solutions (discussed in section 3.4) provide a satisfactory performance, they show several limitations concerning high cost, reduced openness and autonomy.

Considering the prospective applications, some scenarios may require a large number of MIDI connections, and therefore, a large number of Wireless MIDI devices. Furthermore, if Wireless MIDI devices are expensive, it will not be feasible to replace the traditional wired connections with wireless links.

A main decision considering the development of wireless MIDI solutions is the choice of an open or proprietary technology. Solutions supported by proprietary technologies are commonly less flexible and less endowed to improvements than solutions supported on open technologies. Likewise, systems assembled using successful technologies, thus attaining a large customer market, are often cheaper and better supported. Therefore, successful open technologies can enable an initial development of satisfactory solutions and can permit a sustained evolution towards low-cost, high quality products.

Thus, it is the author's belief that, if a cost-effective solution to implement Wireless MIDI devices is to be obtained, it must be based on an open technology used pervasively in large markets such as the information technology.

Musical performances are usually short and last just a few hours at most. However, the logistics involved assume an important role in the success of the performance. If the Wireless MIDI devices have a low autonomy, they must be recharged frequently or replaced by others. Therefore, the direct consequence of low autonomy is an increased logistics complexity. In scenarios with a large number of MIDI instruments this may become unacceptable. The wireless technology chosen must also be adequate for the autonomy requirements of musical performances.

The following sections introduce the key issues of using wireless technologies for supporting MIDI communications by focusing operational and timeliness aspects. In addition, some available commercial products are briefly analyzed.

3.1 Introduction

An alternative commercial solution was presented, in the form of prototype, by MIDIMAN [16], one of the well-known MIDI equipment suppliers, at the NAMM show [17] in 2001 [18]. Figure 3-1 shows a pair of MIDIMAN *Bluetooth* based MIDI prototypes.

This solution was based on the *Bluetooth* technology and offered 4 MIDI channels selectable via 2 DIP switches. However, this solution never reached commercial availability, possibly due to the high cost of the *Bluetooth* module (at the time) or to the poor performance of the prototypes.

The objective of this work is then to assess the possibility of developing a Wireless MIDI solution supported in wireless technologies with potential to become pervasive. Also, as the use of Wireless MIDI is of special interest in stage performances, where a significant number of instruments are concentrated, the solution must be able to support simultaneous transmissions between a reasonably high number of modules (say, at least some tenths).

It is the belief of the author that, to achieve pervasiveness in this application, cost effectiveness must be obtained. Although many wireless technologies could be sought to solve the problem, cost-effectiveness is usually only attained by using COTS (components off the shelf), particularly when those are used in massive markets such as the ones of information technology.

In the sequence of the previous reasoning, and given the envisaged application scenario, only technologies developed for Wireless Personal Area Networks (WPANs) were considered for evaluation (an overview of these technologies is presented in Chapters 4 and 5). At this point in time, *Bluetooth* was considered the most promising solution, mainly due to its predictable low cost, low-power operation, low size and easy handling of the communications between the previewed number of modules within the same space. Another important factor that led to the interest in *Bluetooth* was the possible direct MIDI mapping using the Serial Port Profile (SPP). The SPP offers a serial port replacement mechanism that can be used to replace the original wired connection by a wireless *Bluetooth* based connection. Also, since *Bluetooth* is a well-known technology and people are used to manipulate *Bluetooth* devices, the need for specific training is reduced. However, ZigBee also seemed an interesting technological option despite of the technological lack of availability (at low cost) at the time of this writing.

Figure 3-1 - MIDIMAN Bluetooth-MIDI prototype

The following sections describe the timing requirements for the wireless transmission of MIDI, and the proposed Wireless MIDI architecture. A short overview of the existing commercial solutions is included at the end of the chapter.

3.2 Timing Requirements

As introduced in Chapter 4, the MIDI physical layer uses unidirectional point-to-point connections and therefore the channel is fully available to transmit MIDI messages. So, time is directly coded in the sending instant. Obviously there are delays, but they are small and just related with the transmission of each MIDI command byte. Figure 3-2 shows the transmission of a three-byte MIDI message.

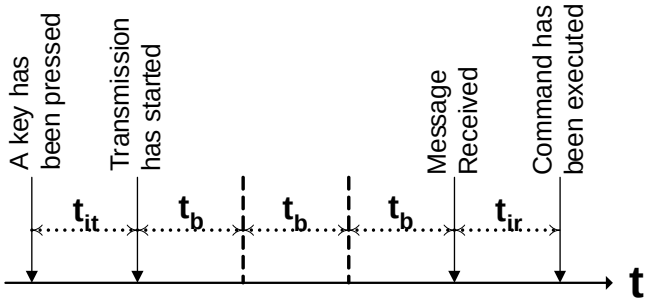


Figure 3-2 - Timeline of a MIDI transmission

It is assumed that the transmitter is able to avoid latency between consecutive byte transmissions. Therefore, the transmission delay t_d of an n-byte MIDI message can be represented by the following equation:

$$t_d = t_{it} + n.t_b + t_{ir} + t_{prop}$$

Where:

t_{it} represents the processing interval at the transmitter

t_b represents the time to transmit a byte

t_{ir} represents the processing interval at the receiver

t_{prop} represents the propagation delay (which can be ignored considering that a 40m cable introduces a delay of approximately 200nsec).

For a three-byte message like a *Note On*, the transmission delay simplifies to:

$$t_d = t_{it} + t_{ir} + 952\mu sec$$

If the two t_i parameters are small (and they usually are) the command will face a delay around 1millisecond before becoming effective at the receiver. Because human hears can perceive sound delays higher than 4 milliseconds [19], these timings are quite appropriate for their discrimination capacity.

Considering a more complex sequence of messages, one must notice that the relative timings between messages are very important. An example can be a chord. A chord is a sequence of keys pressed simultaneously or within a very short interval of time. In this

scenario the produced sound must be heard as a simultaneous set of sounds. The early start or late stop of one of the notes will be easily noticed by the human hear. Figure 3-3 shows the correct MIDI transmission of a three-note chord.

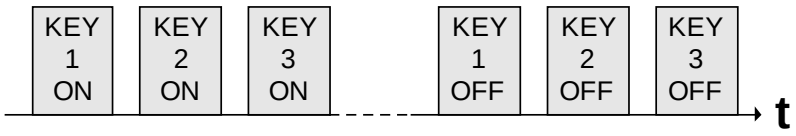


Figure 3-3 - Timeline of a chord

Considering that the delays between *Note On* commands are bounded and that the transmission delay is constant (and also bounded), then the delay between the *Note Off* commands must also be bounded. These considerations guarantee that the chord becomes effective at the receiver without being jeopardized. The overall chord delay will then be a function of the individual byte delays. Figure 3-4 shows the timings involved for a three-note simultaneous key pressure (chord start or chord stop).

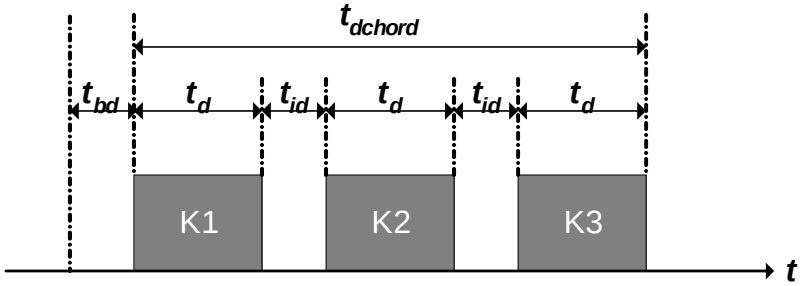


Figure 3-4 - Timeline of a chord start (or stop)

Assuming the theoretical possibility of *k* keys being pressed simultaneously, there will be a small delay t_{db} before the transmission of the first set of bytes corresponding to the first *Note On* (or *Note Off*) command. Besides the command delay t_d there will be another relevant delay between consecutive MIDI commands, here named t_{id}. The sum of these delays is the overall delay experienced by a MIDI chord as expressed in the following equation:

$$t_{dchord}=t_{bd}+k.t_d+(k-1)t_{id}$$

Notice that this approach is valid for the chord start or the chord stop.

Assuming that the transmission is immediate ($t_{db}=0$) after the simultaneous key press and that the delay between consecutive MIDI commands is negligible ($t_{id}=0$), then the overall chord start (or stop) delay will be:

$$t_{dchord}=k.t_d$$

A three-note chord will face a delay of approximately 3 milliseconds, which is acceptable considering the previously discussed boundaries [19].

Despite the tight optimal delay requirements, higher latencies can be tolerated if jitter is bounded to a few milliseconds. Musical performers can tolerate delays of tenths of milliseconds as long as they do not change considerably. This assumption is based on specific field knowledge gathered from field musical experiments with real instruments communicating over MIDI. Therefore, considering that delays can be higher than 4 milliseconds, the system must ensure that their variation will not be noticeable.

3.3 Proposed architecture for Wireless MIDI

As presented in Chapter 2, MIDI connections are unidirectional. Common MIDI devices have at least one input (MIDI In) and two outputs (MIDI Out and MIDI Through). These interfaces allow the connection of several MIDI devices using point-to-point and point-to-multipoint connections. The former are allowed by linking the MIDI Out to the MIDI In interface, while the later by linking the MIDI Out to the MIDI Through interface.

The purpose of connecting several instruments together is to share functionality among MIDI devices. For example, several instruments can be played using a single MIDI device. Figure 3-5 illustrates this concept in which the leftmost MIDI device "plays" the remaining.

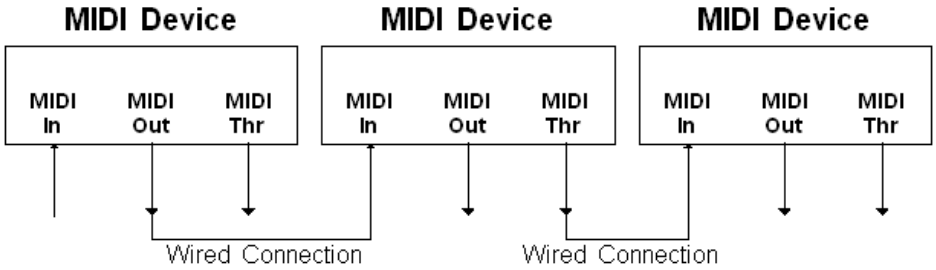


Figure 3-5 - Example of a MIDI setup

The architecture of a MIDI wireless system can be viewed as the simple replacement of the MIDI wired connection with a unidirectional wireless point-to-point connection. In this sense, all the wired connections are replaced with wireless connections (as shown in Figure 3-6).

As it can be seen, each wireless connection links two points in a unidirectional fashion. Therefore, Wireless MIDI (WiMIDI) devices are connected in pairs and the MIDI stream flows in just one direction. For example, WiMIDI IN 01 is connected to WiMIDI OUT 01, thus establishing a unidirectional point-to-point wireless connection between the two attached MIDI devices. Commands sent by the leftmost device are received by the attached

WiMIDI and sent over the air to the corresponding WiMIDI unit (WiMIDI Out 01) that forwards them to the target MIDI device.

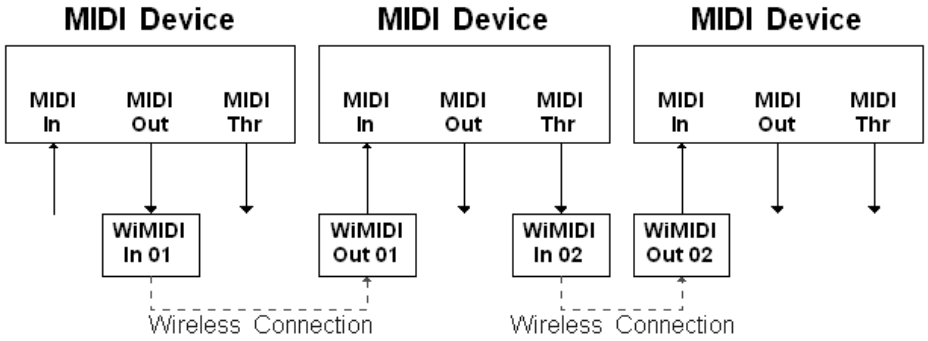


Figure 3-6 - An example of a Wireless MIDI setup

Several solutions could be designed to transmit MIDI wirelessly. However, and to preserve the traditional "look and feel" of establishing connections between MIDI devices, the point-to-point topology was maintained by the architecture of the solution. Obviously, the point-to-point topology is inspired by the MIDI architecture and not by the wireless technology. If other topologies were considered, the number of required WiMIDI nodes could be reduced. For example, if WiMIDI devices allowed point-to-multipoint connections, the network presented in Figure 3-6 could use just 3 devices to communicate the MIDI stream, generated by the leftmost device, to the remaining.

3.4 Commercial Solutions

Nowadays, the market offers some alternatives for supporting Wireless MIDI communications. This section discusses comparable solutions (in terms of cost, range, autonomy and performance) with the proposed WiMIDI system.

The comparative analysis is based on Wireless MIDI adapters, representing renowned Wireless MIDI manufacturers such as Kenton [3], Classical Organ Works [4] and LIMEX [5] that are commercially available (shown in Figure 3-7).

Figure 3-7 - Wireless MIDI commercial solutions

Table 3-1 shows the general characteristics of three different Wireless MIDI products. Clearly, the common characteristics are the high cost and operational range.

Table 3-1 - Comparison between commercial Wireless MIDI products

Product	Cost	Range	Autonomy	Radio	Channels	Weight
LIMEX MIDI	750€	100m	ND*	UHF 902-928 MHz	5	ND*
MIDIJet Pro	400€	150m	30h	ISM 2.4GHz	1	170g
Midistream MIDI	600€	80m	ND*	UHF 869.85MHz	1	ND*

* Not-Defined

The operational range among solutions that use the same radio technology (UHF) is almost identical. However, when a different radio technology is used (ISM), the range almost doubles.

MIDIJet and *Midistream* provide a single channel for communication while *LIMEX* provides five (at the expense of a higher price).

Table 3-1 also demonstrates that manufacturers do not disclose some important details, such as the autonomy. However, the coincidence must be noticed because it takes place for both UHF-based devices. Manufacturers do not provide strict information regarding latency. For example, Kenton says that the devices present "*imperceptible latency and low dropout rate*" while Classic Organ Works just specifies that devices exhibit a "*low latency*".

To sum up, existing solutions are very expensive and largely based on proprietary technology. In addition, their specification does not disclose several important details, such as autonomy and worst-case latency. Furthermore, available solutions do not seem sufficiently feasible to withstand the requirements of the application.

CHAPTER 4

BLUETOOTH OVERVIEW

4.1 Introduction

Bluetooth [20] is an open standard designed for *ad-hoc* short-range wireless networking. This standard operates in the unlicensed ISM 2.4 GHz band and is able to communicate both data and voice. Its main features are: robustness, low complexity, low power consumption and low cost.

Bluetooth is currently at version 1.2 and, since March 2002, the IEEE 802.15 working group has adopted the SIG [21] 1.1 *Bluetooth* specification (without any major changes) and made it an IEEE standard named IEEE 802.15.1 [22].

Bluetooth uses a frequency hopping transceiver in order to lower the signal fading and interference of the wireless channel. This communication channel is divided into time slots and the Time Division Duplex (TDD) scheme is used for transmissions. In this sense,

packets occupy at least one time slot but can be extended to five. These slots can also be reserved for synchronous transmissions. *Bluetooth* supports one asynchronous channel for data and up to three synchronous simultaneous voice channels [23].

The *Bluetooth* protocol supports two types of connections: point-to-point (only two *Bluetooth* units communicate with each other) and point-to-multipoint. In the last form, the channel is shared with all connected *Bluetooth* units. Two or more units sharing the same *Bluetooth* channel form a *piconet* (see Figure 4-1).

The *piconet* is the atomic network of *Bluetooth*. Each *piconet* has one *piconet* master and one (or more) *piconet* slave(s) (seven active slaves at most). However, several slaves may be attached to the network in a so-called parked state. These parked slaves are not active, but synchronized with the *piconet* master.

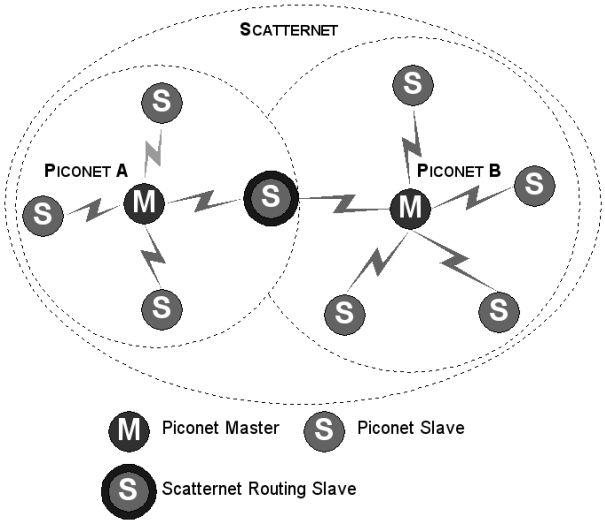


Figure 4-1 - Bluetooth scatternet.

Piconets covering common areas form a scatternet. *Bluetooth* units that belong to one *piconet* may participate in other *piconets* by using a Time Division Multiplex (TDM) scheme. Moreover, a *piconet* master may be a *piconet* slave in a neighbor *piconet*. *Piconets* are not frequency hop synchronized, e.g., each one has its own specific hop sequence. This partially avoids *piconet* interference.

4.2 Stack

The *Bluetooth* specification defines not only the physical medium (radio interface), but also a complete communication stack (as seen in Figure 4-2). Besides defining different communication protocols the stack includes the specification of protocols to allow devices to find their neighbors and to advertise their services.

The *Bluetooth* stack can be divided into three logical groups of protocols [24]: the application protocol group, the middleware protocol group and the transport protocol group.

The application protocol group comprises the applications (*Bluetooth*-aware or not) that use the *Bluetooth* technology.

The middleware protocol group consists on both *Bluetooth* specific protocols like the serial port emulation (RFCOMM) and other adopted protocols like the Object Exchange Protocol (OBEX).

The transport protocol group is composed of protocols exclusively developed for the *Bluetooth* technology, like the Logical Link Control and Adaptation Protocol (L2CAP) or the Host Controller Interface (HCI).

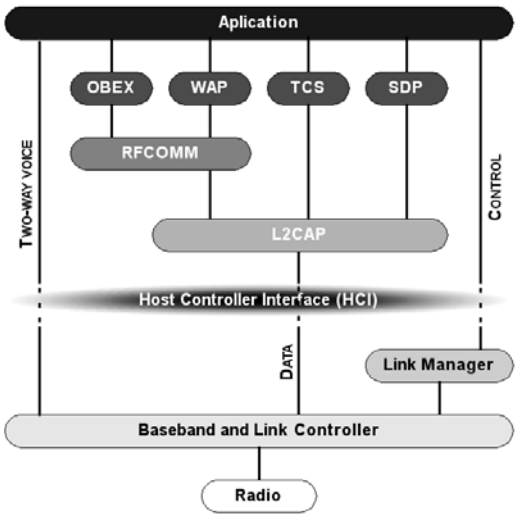


Figure 4-2 - Bluetooth protocol stack

A short discussion of the protocols defined for the different layers of the *Bluetooth* stack is presented below.

4.2.1 Radio

As pointed, the *Bluetooth* radio operates in the unlicensed ISM (Industrial, Scientific and Medical) band at 2.4 GHz. The antenna radiation is classified in one of the three power classes:

- Power Class 1: designed for long range (~100m) devices, with a maximum output power of 100 mW;

- Power Class 2: for ordinary range (~10m) devices, with a maximum output power of 2.5 mW;
- Power Class 3: for short-range (~10cm) devices, with a maximum output power of 1 mW.

Data is transmitted at a maximum raw rate of 1Mbit/s. However, due to communications overhead, only a maximum of 721Kbit/s is (actually) achievable.

Bluetooth uses 79[2] hop frequencies spaced 1 MHz apart ranging from 2.402 MHz to 2.480 MHz. The hop rate is 1600 hops per second and each is separated from the following by a 625μs time window.

A *piconet* is characterized by a specific hopping pattern determined by the *piconet* master ID and its clock. The overall hopping pattern is divided in 32[3] hop segments.

The *Bluetooth* radio uses GFSK (Gaussian Frequency Shift Keying) for modulation by representing a binary one by a positive frequency deviation and a binary zero by a negative frequency deviation.

The radio tolerance for transmitting is ±75Hz from the specified central frequency Fc, meaning that the deviation from Fc must be lower than 75Hz.

4.2.2 Baseband

The *Bluetooth* Baseband packet follows the general packet structure shown in Figure 4-3.

Figure 4-3 - Baseband packet structure

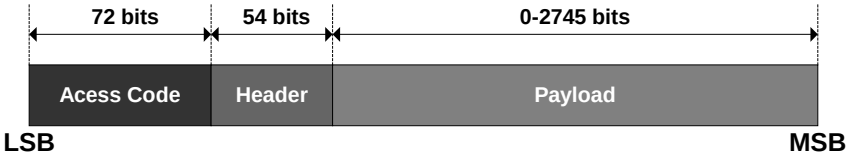


Figure 4-3 - Baseband packet structure

[2] In a few countries (e.g., France) this frequency band range is (temporarily) reduced, and a 23-hop system is used instead.

[3] In reduced hop systems (23 hops) the hoping pattern is divided in 16 segments.

This packet structure contains an Access Code, a Header and a Payload. The Access Code field is 72-bit wide and is used for synchronization. However, depending on the application context, it may contain the *piconet* identity or the address of the recipient.

The Header field is 54-bit wide and includes the destination address, the type of payload that follows and error control information.

The payload field is variable in length and includes the message to be transmitted.

As pointed, the communication channel is divided in time slots, each one occupying 625μs. These slots are numbered according to the master clock of the *piconet*. The master transmits on even numbered slots and the slave in odd numbered slots. Each transmission takes place at one new hopping frequency, and a complete data packet is sent in each slot. This means that there is no frequency shift before the end of the packet transmission, even if the packet occupies more than one slot. Figure 4-4 unveils the communication process for a single slave *piconet*.

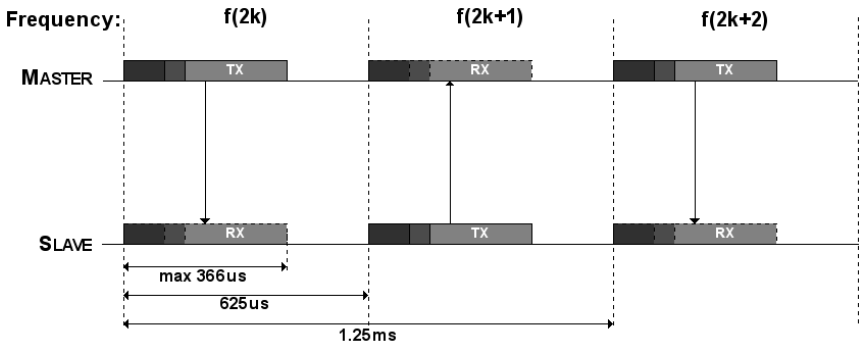


Figure 4-4 - Communication process for single-slave operation

Point-to-multipoint communication occurs when a *piconet* contains more than one slave. In this scenario, and to prevent *piconet* members of jamming each other with simultaneous transmissions, a *Time Division Duplex* (TDD) scheme is used. This means that slaves multiplex the overall throughput. Figure 4-5 demonstrates the procedure.

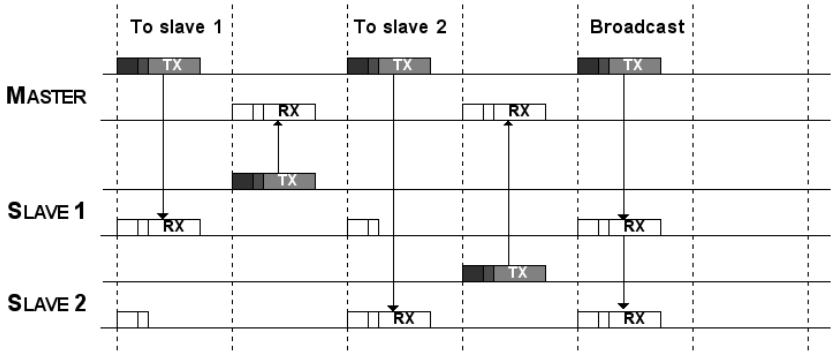


Figure 4-5 - Communication process for multi-slave operation

As described, the master transmits on even-numbered slots and the slaves on odd-numbered slots. However, slaves can only transmit when directly addressed by the master in the previous time slot. In this sense, if the master sends a packet to slave 1, slave 2 will remain with its receiver turned on only while decoding the packet access code (identifying the *piconet*) and header (identifying the destination). Afterwards, it will realize that the packet is not addressed to it and will turn its receiver off until the next even-numbered slot.

As can be seen in Figure 4-5, the *piconet* slaves only communicate with the *piconet* master. This means that slave-to-slave data transfers are only possible through the master or by creating a separate *piconet* through which they can communicate directly (one being the *piconet* master and the other the *piconet* slave).

The support of multi-slot packets enables the increase of the achieved data throughput. Multi-slot packets can either have a duration of three or five time slots. Figure 4-6 depicts the timing of one-slot, three-slot and five-slot packets.

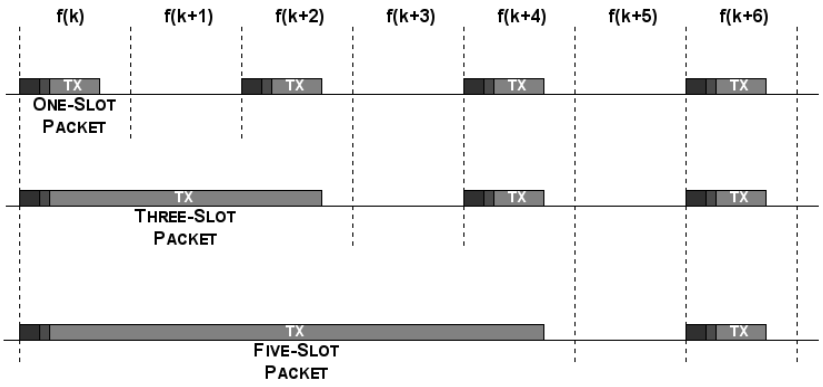


Figure 4-6 - Timings for one, three, and five slot packets

As represented in Figure 4-6, packets are sent in a single hop frequency independently of their length.

4.2.2.1 Links

Bluetooth devices can establish two[4] different logical transports: asynchronous connectionless (ACL) and synchronous connection-oriented (SCO) links. ACL links are used for data communication while SCO links are used for real-time two-way voice [25]. Applications with low latency and low integrity constraints use SCO links. An SCO link is a circuit-switched, point-to-point link between a master and a single slave. The low latency characteristic is achieved through well-timed transmissions in specific slots and with the absence of retransmission mechanisms. However, the lack of retransmission mechanisms can lead to loss of data when transmission failures occur. This is not a common problem since, for example, voice reproduction from a digitized bit stream can tolerate a fairly high percentage of bit errors.

SCO Packets are exchanged in pairs, firstly from the master to the slave and afterwards (next slot), in the opposite direction. Slaves are allowed to transmit in their reserved time slots, even if the master did not transmit in the previous slot. However, if the master transmits a higher priority packet in the reserved slot, the slave is not allowed to transmit. This means that, in case of reserved slot dispute, the master always wins the right to transmit.

When data integrity is more important than latency, ACL links are used. An ACL link is a packet-switched link between a master and a slave. When a packet is received with uncorrectable errors it is retransmitted until proper reception. Of course, the number of retransmissions increases directly with the bit error rate of the channel (BER) leading to high latency times.

A *piconet* master can transmit broadcast packets that will be received by all slaves. The slaves are not permitted to reply in the slave-to-master slot following a broadcast packet.

ACL links can also be used to carry isochronous data. Isochronous data is less time critical than real-time two-way voice. Streaming audio applications can use ACL links for buffered audio transmission. Retransmissions are possible without affecting considerably the audio reproduction. However, if the BER of the channel is high, it may be necessary to

[4] In fact, the *Bluetooth* Specification Version 1.2 introduces another logical transport named *extended* SCO (eSCO). This logical transport is one of the main improvements introduced by the latest *Bluetooth* specification and will be described later in section 4.3.

flush periodically bad received packets, which can result in audible sound quality degradation.

4.2.2.2 Link Controller

The Link Control (LC) sublayer resides within the Baseband layer. It is responsible for managing the discoverability of the device, establishing connections and maintaining them.

Aiming to establish *piconets*, inquiry and page procedures are defined. Prior to the *piconet* creation, the devices involved are not yet master and slave. To reduce ambiguity, they are called prospective master (p-master) and prospective slave (p-slave).

In the inquiry phase the p-master discovers other *Bluetooth* devices that are in range by broadcasting an inquiry message. The p-slaves that receive[5] the inquiry message return an FHS (Frequency-Hop Synchronization) packet including, among other information, their identity and clock information. If all p-slaves reply simultaneously their messages will collide. To resolve the FHS collisions the p-slaves reply using a back off mechanism.

In the page phase the p-master tries to establish a *Bluetooth* connection with a particular p-slave (that may have been discovered during the inquiry phase). The p-slave must be connectable, which means that it must be in the page scan state to accept a connection request. However, since one of the *Bluetooth* requirements is low power operation, a p-slave is most of the time in a sleep state. When awaken from this state, a p-slave performs a page scan at a different hop frequency for a short period of time. If this time is sufficiently long a p-master will eventually send a page at this frequency and the connection establishment may proceed. However, if a p-slave is awake for a long period of time it will drain out the batteries rapidly. A trade-off has to be made between response time and power consumption. The p-master transmits the access code repeatedly at different frequencies. In a 10-millisecond period 16 hop frequencies are paged (in each 1.25ms two access codes are transmitted at two different hop frequencies). If once awaken, an idle p-slave performs a page scan in any of these 16 frequencies, it will receive the access code required to start the Baseband connection procedure. In this sense, the p-slave will start by notifying the p-master of the received access code. The p-master will transmit an FHS packet containing all the required information to start the connection. This information is then used by both, p-master and p-slave, to establish the *piconet*. After a successful Baseband link establishment, both units can exchange roles if they wish (the slave may become master and vice-versa). It must noticed that, if a device does not make itself discoverable by being in the inquiry scan state for a determined amount of time, it cannot be found by neighbor devices (unless they already know the address of the device).

[5] p-slaves are only able to listen inquiries when operating in inquiry scan.

Also, if a device does not make itself connectable by being in the page scan state for a determined amount of time, neighbor devices will not be able to connect with it.

Additional information concerning the connection establishment can be found in [26] and [27].

4.2.3 LMP

The Link Manager (LM) translates the higher layer commands into operations at the Baseband level. Its main functions are: link configuration and information, and *piconet* management and security. The link configuration and information functions are especially important when, after a successful paging procedure, one master and one slave form a *piconet*. Both need to discover what link features (multi-slot support, RSSI[6], etc.) are available on the other end. These functions also play an important role for setting QoS, power control and other configuration options during the time a connection is active.

The *piconet* management functions include the management of slave attachment and detachment, master-slave role switch, SCO link establishment and handling of low-power modes (sniff, hold and park). These low-power modes are not discussed here for extension considerations.

The security management functions handle most of the implementations associated with the authentication and/or encryption of the *Bluetooth* link.

The LM communicates with its peers on other *Bluetooth* devices by using the Link Management Protocol (LMP). Figure 4-7 depicts the payload body of an LMP Packet Data Unit (PDU). The payload body contains a transaction identifier (TID), an *opcode* and a variable number of parameters. The first bit to be transmitted is the 1-bit TID. When the PDU is originated in the master the TID bit is set to 0. When the PDU is originated in the slave the TID is set to 1. This bit is commonly used to track which node started the parameter negotiation.

The *opcode* is a 7-bit PDU designator. An *opcode* identifies a specific command that may have (or not) several parameters being provided after the *opcode*. If a parameter exists, it can be as short as one byte or as long as 16 bytes (usually parameters occupy an integer number of bytes).

[6] The Receiver Signal Strength Indicator (RSSI), as the name implies, is a parameter used to measure the receiver signal strength of a *Bluetooth* device. It is used to evaluate if the transmitter on the other side of the link should increase (or decrease) its output power level in order to maintain the desired level of connectivity using the less possible power.

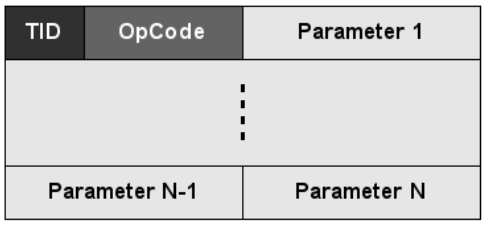


Figure 4-7 - LMP PDU payload body

As pointed, the LMP provides mechanisms for encryption mode negotiation and for encryption key coordination between both peers. LMP also supports messages for QoS configuration. In this sense, the used packet types can change to reflect changes in the BER of the channel. For example, when a link is firstly set-up it uses single slot packets. However, the use of multi-slot packets increases the overall throughput of the link. Provided that the BER is sufficiently low to support low latency communications, multi-slot packets can be used to increase the throughput of the link.

4.2.4 L2CAP

The Logical Link Control and Adaptation Protocol (L2CAP) is a protocol layer that provides connection-oriented (master to slave and slave to master) and connectionless (master to multiple slaves - broadcast) data services to upper layer protocols. Although L2CAP does not provide real-time communication capabilities as SCO links, it may communicate with the LM to assist the channel setup. L2CAP supports two types of user data: user asynchronous (UA) and user isochronous (UI). The UA type offers reliability guarantees by forcing the retransmission of data until successfully received. As pointed before, if the BER of the channel is high, large latency times will occur. In the UI type, data reliability is as important as latency times. In this sense, latency can supersede reliability requirements (multimedia streaming is an example). UA and UI data types are supported in a specific logical link named ACL-U, which allows the user data to be framed and transmitted according to its nature (asynchronous or isochronous).

The L2CAP functions can be divided into four categories: protocol multiplexing, packet segmentation and reassembly, QoS and group management.

The protocol multiplexing function ensures the sharing of an ACL connection with several higher layer links. In this sense, it allows different user applications to run on a single ACL link. The protocol differentiation is achieved through the use of channel numbers (labels) that guarantee proper routing of ACL payloads belonging to different higher layer protocols.

L2CAP transparently provides protocol multiplexing to higher layers.

The segmentation and reassembly function ensures that higher layer packets with large (sometimes as long as 64K bytes) Maximum Transmission Units (MTUs) are segmented into smaller *Bluetooth* Baseband packets for proper transmission. It also ensures the opposite packet reassembly by combining small *Bluetooth* Baseband packets into larger higher layer packets. This process is transparent for higher layers.

The QoS function allows the implementation of a QoS level for each protocol in terms of parameters such as bandwidth, latency, and delay variation. These QoS settings are set using the token bucket traffic model. However, by default, the QoS level is set to best effort, meaning that it will perform the best it can under any circumstances.

The group management function arises from the fact that several higher layer protocols require the capability of managing address groups. The *Bluetooth* LM supports a group of devices named *piconet* that includes all active and parked slaves. L2CAP takes this concept into the next level by allowing the mapping of protocol groups within the *piconet*. An example could be a real-time streaming video transmission for two active slaves within a seven-slave *piconet*.

4.2.5 RFCOMM

The RFCOMM Protocol is used "*to expose a serial interface to the packet-based Bluetooth transport layers*" [28]. This protocol emulates standard RS232 control and data signaling over the *Bluetooth* Baseband. It is based on the ETSI 07.10 standard [29] and supports the emulation and multiplexing of several (maximum 30) serial links over a single connection. The ETSI 07.10 standard is an asymmetric protocol used by GSM cellular phones to multiplex several streams of data onto one physical serial cable.

Legacy applications designed to operate over serial cables can run on top of a *Bluetooth* link without significant modifications using the RFCOMM protocol. Most of the actual applications developed for *Bluetooth* use the RFCOMM protocol as part of their stack.

4.2.6 SDP

The Service Discovery Protocol arose from the original envisioned rich operating space for *Bluetooth* (a pervasive world offering different services). In this sense, a large variety of services would be available to choose from. Using SDP, a *Bluetooth* unit can inquire about the services that another *Bluetooth* unit offers and learn how to access them. This means that SDP does not provide the services, but information about them.

Services and attributes are described by universally unique identifiers (UUIDs). Usually UUIDs are 128 bits long, but for known services, 16-bit and 32-bit UUIDs may also be used.

The following section briefly describes some of the most important improvements introduced in the latest *Bluetooth* specifications.

4.3 Recent Improvements

In November 2004 the *Bluetooth* SIG launched the new *Bluetooth* Core Specification Version 2.0 + Enhanced Data Rate (EDR) [30]. This specification promises several improvements to the previous versions, namely:

- Increased data rate (up to ten times in specific scenarios);
- Lower power consumption achieved by decreasing duty cycles;
- Simplified multi-link scenarios through the use of additional bandwidth;
- Improved Bit Error Rate (BER) performance;
- Backward compatibility.

Three major companies, Cambridge Silicon Radio (CSR) [31], Broadcom [32] and RF Micro Devices (RFMD) [33], support this *Bluetooth* specification. CSR implements its BlueCore4 chip with 0.18µm technology, while Broadcom and RFMD (BCM2045 and SiW4000 chips respectively) use 0.13µm technology in their implementation.

At the time of this writing only *Bluetooth* Specification 1.2 is available for non-SIG members. Therefore, the following analysis is solely based in the *Bluetooth* Specification Version 1.2 [34] [35].

This release improves several existing features to achieve shorter connection times, higher quality audio links and improved co-existence with other wireless technologies, namely Wi-Fi [36]. A group of four new features called *Faster Connections* has been introduced to shorten the connection establishment. The *enhanced inquiry* is part of this group and with the interlaced *inquiry scan* and *page scan* can reduce up to half the connection times (inquiry + page). The *RSSI with inquiry results* is the last member of this group and is targeted to devices with limited display capabilities. It acts by ordering the inquiry results by RSSI value from the highest (which should be the desired) to the lowest. So, the device will show to the user a list of available devices, ranging from the nearest to the farthest.

Traditionally, audio is transmitted over SCO links. Because these links do not have error detection capabilities, sometimes users experience data loss in noisy environments. To overcome this problem *Bluetooth* Specification Version 1.2 introduces an *extended* SCO (eSCO) logical transport. When an eSCO transport is established the *piconet* master assigns an additional address (LT_ADDR) to the slave. This address is used in all eSCO traffic allowing the separation of the eSCO Automatic Repeat reQuest (ARQ) scheme from

the ACL ARQ scheme. The eSCO transport supports error-detection and limited retransmissions. These features contribute to an improved audio quality.

The co-existence with other wireless technologies, a long-term issue, was addressed by introducing a mechanism known as *Adaptative Frequency Hopping* (AFH). This mechanism works by reducing the nominal number of hopping channels according to their occupation by other ISM technologies such as Wi-Fi, microwave ovens, cordless telephones, etc. This way, channels in use are marked as occupied and are not considered in the hopping pattern. *Bluetooth* Specification Version 1.2 defines AFH as being mandatory. This specification globally describes the AFH mechanism. However, it does not specify the channel occupation assessment algorithm. This means that manufacturers are free to implement their own solution, which may lead to different performances (in noisy environments) among different manufacturers.

The following section describes the role of the *Bluetooth* profiles within the Specification.

4.4 Profiles

Bluetooth profiles define the implementation of *Bluetooth* usage models. These models refer to specific arrangements of protocols within the *Bluetooth* specification. *Bluetooth* profiles have three main purposes [25]:

- To reduce the number of options by setting parameter ranges within the used protocols;
- To specify the order in which procedures are combined;
- To provide similar user experience in *Bluetooth* devices of different manufacturers.

A *Bluetooth* profile can be seen as a "*vertical slice in the Bluetooth protocol stack*" [25]. In this sense, it only "selects" particular functionalities from each layer according to the specific *Bluetooth* function being developed. As illustrated in Figure 4-8, *Bluetooth* profiles are organized into groups. Each group inherits features from the one beneath, from which it is built upon [37]. Additional features from the standard are gathered as required. This scheme allows recycling features among profiles and reduces the costs concerning the development cycle.

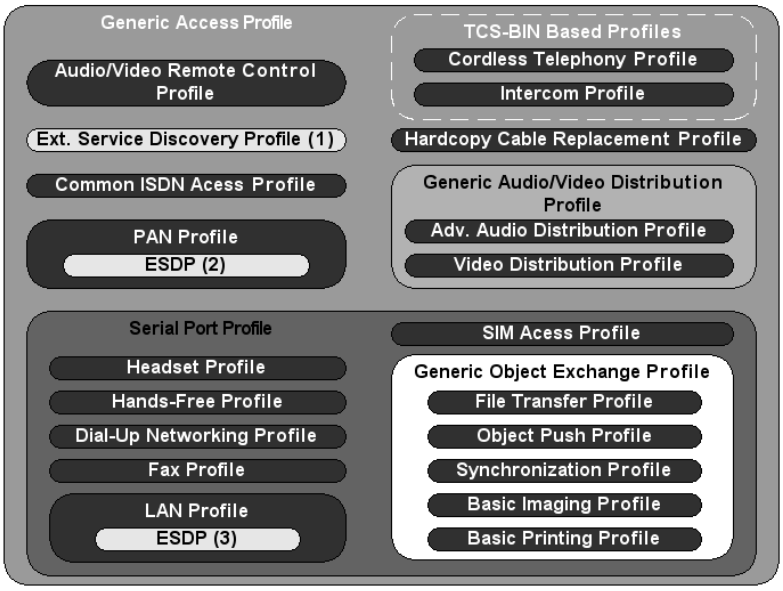


Figure 4-8 - Bluetooth profiles

Bluetooth profiles were drawn from typical usage scenarios (use cases) and grouped together according to shared elements [38]. However, there are profiles developed for different purposes than just the mapping of a particular usage scenario. These profiles are usually referred to as Generic Profiles and are the Generic Access Profile (GAP) and the Service Discovery Application Profile (SDAP). The former is the elemental profile, from which all remaining are built upon. Its main purpose is to ensure that all devices can successfully establish a Baseband link. The latter "*provides a common and standard method for performing service discovery using the Bluetooth protocol stack*" [37]. The Generic Access Profile, for example, defines several elements that must be taken into account when developing an application. Not only it defines the requirements for features that must be implemented in all devices, but it also specifies requirements regarding the discoverability, connectivity and security of *Bluetooth* devices. Additionally, this profile defines a specific terminology to be used in the user interface. It can, however, be different from the one used through the *Bluetooth* Specification.

Another important profile is the Serial Port Profile (SPP). This is probably the most popular *Bluetooth* profile (in parallel with the Headset Profile) since it addresses one of the initial *Bluetooth* target applications (cable replacement). This profile is based on the GSM 07.10 standard [29] and allows multiplexing several serial connections over a single serial link. It accomplishes this task by specifying the protocols and procedures required to seamlessly emulate (using *Bluetooth*) a RS232 wired connection. In terms of hierarchical structure, the SPP is built upon the GAP (see Figure 4-8) and the additional features are drawn in from the *Bluetooth* standard.

Resuming, *Bluetooth* profiles have the purpose of guaranteeing interoperability and similar user experience when using equipment from different manufacturers.

4.5 Conclusion

This chapter provided an overview of the *Bluetooth* protocol by focusing on its layered structure. This technology offers several features that seem suitable to be used in the envisioned application. The most important are the low-cost, the technological availability, the adequate operating range and the noise immunity. Besides providing several key features, *Bluetooth* is an evolving technology that seems a good option for supporting wireless communications. This means that, even if its performance is not as good as others (at present), there is enough room to improvement and thus it can be considered a promising solution to implement wireless MIDI.

CHAPTER 5

WIRELESS PERSONAL AREA NETWORKS – AN OVERVIEW

5.1 WI-FI

The IEEE 802.11 [39] is the leading standard in wireless LANs. The original scope of this standard was “to develop a Medium Access Control (MAC) and Physical Layer (PHY) specification for wireless connectivity for fixed, portable and moving stations within a local area” [40]. After its release in 1997, the IEEE 802.11 protocol was ratified in 1999 to support data rates above the 10 Mbit/s barrier. In 2003 the IEEE 802.11 standard was amended to further extend its data rate to 54 Mbit/s within the 2.4 GHz band. This evolution has been motivated by the increasing demand for higher data rates. However, the original goal still remains.

5.1.1 Overview

The IEEE 802.11 standard specifies the MAC and PHY layers for Wireless Local Area Networks (WLANs). This standard adopts the IEEE 802 standard (802.2) for the Logic Link Layer (LLC), which, with the MAC sub-layer, forms the Open Systems Interconnect (OSI) Data Link Layer (DLL). The 802.11 PHY specifies two distinct physical layers: Direct Sequence Spread Spectrum (DSSS) and Frequency Hopping Spread Spectrum (FHSS). As introduced, several revisions to the original standard have been made. Considering the last revision (2003), the transmission technology and the operating spectrum, the 802.11 protocol can be classified in three categories [41]:

- 802.11a (Orthogonal Frequency-Division Multiplexing (OFDM), 5 GHz);
- 802.11b (High-Rate DSSS (HR/DSSS), 2.4 GHz) and
- 802.11g (OFDM, 2.4 GHz).

The IEEE 802.11a (rev. 1999) standard is based on a multicarrier technique named OFDM and operates in the unlicensed national information infrastructure (U-NII) band (USA). This standard supports data rates between 6 and 54 Mbit/s. The IEEE 802.11b (rev. 1999) operates in the 2.4 GHz Industrial, Scientific and Medical Band (ISM) using a High Rate DSSS (HR/DSSS) technique for achieving data rates between 1 and 11 Mbit/s. Finally, the IEEE 802.11g (rev. 2003) standard also operates in the ISM band and is able to support data rates up to 54 Mbit/s using the OFDM multicarrier technique. Since 802.11g and 802.11b use the same operating spectrum their compatibility is guaranteed, from a bandwidth perspective.

The IEEE 802.11 MAC is considered the key to the 802.11 specification [42]. It links the PHY layer with higher layers providing core framing operations and interaction with (possibly) wired backbones. Following the success path of Ethernet, it uses a Carrier Sense Multiple Access (CSMA) scheme to control the medium access. However, it does not use a Collision Detection (CD) mechanism as Ethernet. Instead, it uses Collision Avoidance (CA) thus reducing the waste of valuable transmission capacity.

The following subsections present a brief overview of the basic components and supported network types that characterize the IEEE 802.11 protocol.

5.1.2 Components and architecture

802.11 networks are built on four physical components: Distribution System (DS), Access Points (AP), Wireless medium and stations. These components are related as shown in Figure 5-1.

The DS is the interface that links stations within a WLAN to the exterior world (usually the Web) by forwarding frames to their destinations. APs convert frames in the 802.11 format into different frame formats allowing them to be transmitted in different communication media (such as Ethernet for example). The IEEE 802.11 protocol uses a wireless medium as the frame communication support. This wireless medium is based on two PHYs: Radio Frequency (RF) and infrared. The RF PHY has become very popular, mostly because of its improved operating range and performance.

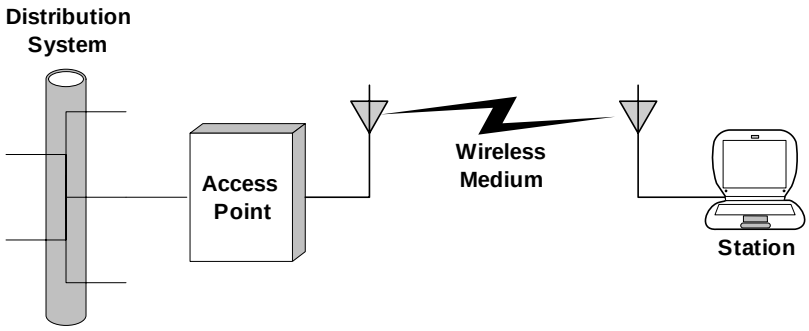


Figure 5-1 - IEEE 802.11 components

Stations are computing devices that include a wireless network interface. These computing devices can be laptops, PDAs, PCs or other micro-controller based devices.

5.1.2.1 Types of Networks

The *Basic Service Set* (BSS) is a group of stations that communicate with each other. Figure 5-2 shows the two forms a BSS can take: independent and infrastructure.

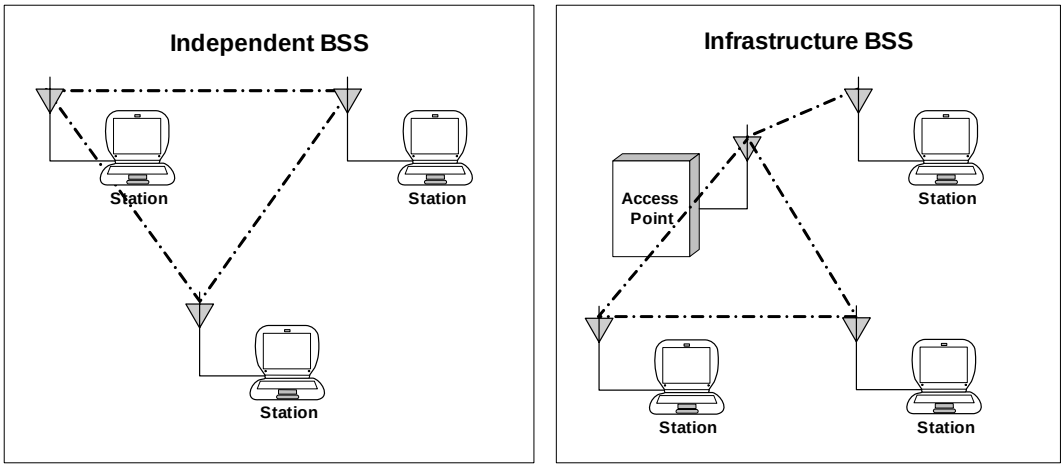


Figure 5-2 - IEEE 802.11 supported networks

In the Independent BSS form (IBSS), stations can communicate directly with each other providing that they are in range. This form is also known as *ad-hoc* BSS and, in its

smallest arrangement, it only includes two stations. Usually, IBSSs are built to provide connectivity within a small area to a small number of stations for a short period of time (conference meetings, spontaneous data exchange, etc.).

Infrastructure networks use an access point to mediate communication. When a station initiates a frame transfer to another station, it must transfer the frames to the AP, which in turn communicates them to the target station (even if the target station is in range). Although the communication needs two hops, two advantages arise immediately from this procedure:

- The operating range limit of an Infrastructure BSS is determined by the AP coverage. In this sense, even if two stations are not in range from each other, they may communicate provided that they are within the reach of the AP. It seems obvious that allowing direct communication between stations can lead to a higher throughput. However, this would increase the PHY complexity since it would require the embodiment of additional mechanisms to manage neighbor relationships within the service area.
- Most stations requiring wireless communications are mobile, and thus battery powered. APs are in a good position (Infrastructure BSS) to assist mobile stations that require power savings. If an AP acknowledges a mobile station entering in a power saving mode, it may operate by buffering all frames and only delivering them when requested.

The following sub-sections provide an overview of the IEEE 802.11 MAC and PHY layers.

5.1.2.2 MAC layer

In the 802.11 MAC, the access to the wireless medium is controlled by coordination functions. Therefore, the IEEE 802.11 MAC defines two coordination functions: Distributed Coordination Function (DCF) and Point Coordination Function (PCF) (as shown in Figure 5-3). The former provides an Ethernet-like CSMA/CA medium access while the latter provides a contention-free medium access.

The CSMA/CA access mechanism works by sensing the medium for transmissions. If the medium is found busy (another device might be transmitting), then the station will postpone its transmission to a later time. Otherwise, the station is allowed to transmit. Furthermore, collisions might occur if two stations sense the medium at the same time and decide to transmit simultaneously.

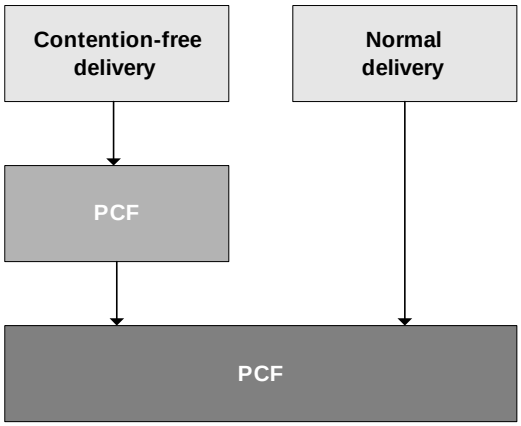


Figure 5-3 - IEEE 802.11 MAC coordination functions

The collision avoidance (CA) mechanism works together with a positive acknowledgment scheme to avoid collisions. In this sense, if a station senses the medium as occupied then it defers its transmission to a later time, once the medium has been free for a specified amount of time (called Distributed Inter-Frame Space - DIFS), the station transmits. The receiving station will check the Cyclic Redundancy Check (CRC) packet field and will send an Acknowledgment (ACK) packet notifying the sender of a successful transmission. Of course, if the sender does not receive the ACK packet it means that a collision has occurred and it must retransmit the packet until it is properly acknowledged. To further reduce the probability of collisions a CTS/RTS clearing technique is used, partially avoiding collisions due to hidden nodes [43]. This technique is defined, in the standard, within the Virtual Carrier Sense mechanism. A station willing to transmit will first transmit a short control packet called Request to Send (RTS), which includes some information (source, destination and duration) about the following transaction. If the medium is available for transmission, the receiving station will respond with a Clear to Send (CTS) packet indicating that the sender is allowed to start its transmission. Notice that the transmission duration, present in both control packets RTS and CTS, accounts for the ACK control packet. All stations receiving these control packets (RST and CTS) must update their Virtual Sense Indicators (defined as Network Allocation Vector - NAV) for the given transaction duration and use them together with the Physical Carrier Sense mechanism. Additionally, an exponential backoff algorithm is used in several scenarios: when a station senses an occupied medium before a packet transmission, after each retransmission and finally after a successful transmission. When a station wishes to transmit and the medium has been available for at least the DIFS time the exponential backoff algorithm is not used.

Although PCF seems more interesting since it implies that there is no need for contention it can only be used in infrastructure networks. In addition, the loose PCF specification leaves several issues unresolved [41]:

- PCF experiences substantial delay at low loads; this occurs because stations must always wait for polling, even if the wireless channel is idle;
- Because the AP needs to contend for the channel using the DCF at the beginning of the contention-free period (CFP), the actual period of contention-free polling may vary;
- In scenarios with a large number of streams, the AP will have an enormous difficulty in managing the polling procedure without affecting other applications using DCF contention;
- PCF is a centralized medium access mechanism controlled by the point coordinator. If a failure occurs, all the associated stations will not able to access the channel and therefore it will not be able to communicate.

These drawbacks justify the general lack of interest in the PCF scheme. Moreover, the research community and the industry directed their research efforts to DCF due to its distributed nature.

IEEE 802.11 MAC Frame Format

Figure 5-4 shows the format of a general 802.11 MAC frame. The 802.11 MAC frame differs from the Ethernet frame mostly due to the lack of the type/length and preamble fields. The preamble field is part of the physical layer, and the type and length fields are present in the header of the 802.11 frame.

The first field of a MAC frame is the Frame Control. This field holds information about the MAC protocol version, the type of MAC frame and additional control information (fragmentation, retransmission, power management, encryption, bit ordering, etc.).

Afterwards, is the Duration/ID field, which can be interpreted in three different ways according to the two most significant bits. When the most significant bit (MSB) is zero the duration/ID field is used to set the amount of time that must elapse until the current transmission session is complete (and the channel can be sensed again). This amount of time is usually known as Network Allocation Vector (NAV). During the CFP the MSB is 1, the second MSB is 0 and remaining bits are 0. In this scenario the duration/ID field is interpreted as the NAV. Therefore, it allows any stations that did not receive the Beacon announcing the CFP to update their NAV with a rather large value so that they do not interfere with current contention-free transmissions.

When stations require a power saving mode they wake-up periodically and send a Power Save (PS) poll frame (PS-Poll) indicating the BSS they belong to. These PS-Poll frames are characterized by having the two MSB bits of the duration/ID field equal to 0. The

addressed AP receives the PS-Poll frame and replies with the buffered frames (if they exist).

The following fields are addresses. An IEEE MAC frame may contain a maximum of four addresses. Figure 5-4 illustrates these numbered addresses once they vary depending on the frame type. The IEEE 802.11 addressing follows the conventions used for the other IEEE 802 networks (such as Ethernet for example). The address fields, if present, contain one of the following 48-bit IEEE 802 addresses: destination address, source address, receiver address, transmitter address, *Basic Service Set ID* (BSSID). In infrastructure networks, the BSSID is the MAC address used by the wireless interface in the access point.

The sequence control field holds the Sequence Number and the Fragment Number subfields. This field is used for both defragmentation and discarding duplicate frames.

The Frame Body, also known as Data field, carries the higher-layer payload. The IEEE 802.11 standard specifies that the maximum payload length is 2,304 bytes. However, implementations must support frame bodies of 2,312 bytes to accommodate the Wired Equivalent Privacy (WEP) overhead.

The last field is the Frame Check Sequence (FCS). The FCS allows stations to verify the integrity of the frames and is performed over all fields in the MAC header and body.

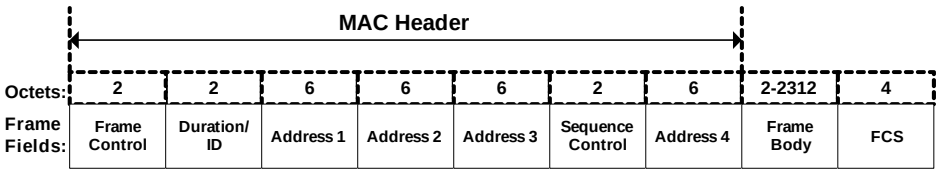


Figure 5-4 - IEEE 802.11 MAC frame format

5.1.2.3 PHY layer

The IEEE 802.11 PHY is divided into two sub-layers (Figure 5-5): the *Physical Layer Convergence Procedure* (PLCP) sub-layer and the *Physical Medium Dependent* (PMD) sub-layer.

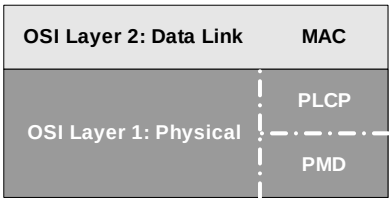


Figure 5-5 - PHY logical architecture

The PLCP sub-layer provides a Service Access Point (SAP) to the MAC layer from which it can communicate with the Physical Layer. The PMD, as the name implies, interfaces directly with the transmission medium by providing the actual transmission and reception of data.

The following sub-section focuses on the PLCP sub-layer of the IEEE 802.11b PHY. This PHY has been chosen due to its popularity when compared to the IEEE 802.11g PHY.

HR/DSSS PHY

The 802.11b PHY, also known as the high-rate, direct-sequence PHY (HR/DSSS) uses the same channels as the original low-rate direct sequence PHY.

The IEEE 802.11 working group has chosen a different encoding method known as Complementary Code Keying (CCK), to increase the number of encoded bits per symbol and, therefore, to increase the data throughput to 5.5 Mbps or 11 Mbps.

As pointed before the HR/DSSS PHY is split into two parts: PLCP and PMD. The IEEE 802.11b PLCP sub-layer supports two types of frames as shown in Figure 5-6. The “long” frame format must be supported for backward compatibility with Direct Sequence (DS) PLCPs. However, an optional “short” PLCP format may be used for increased efficiency and improved throughput.

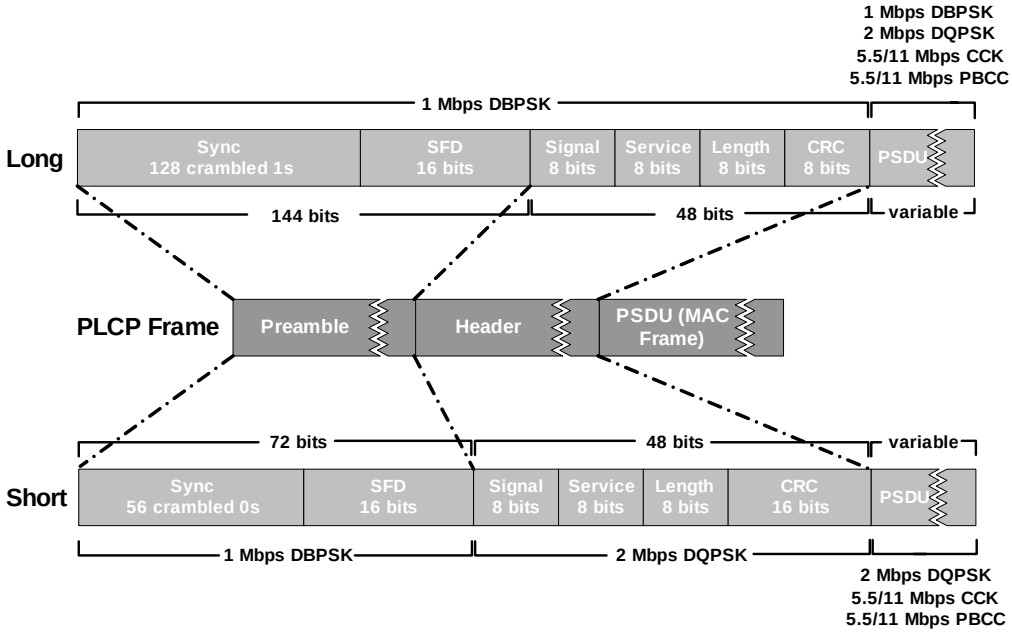


Figure 5-6 - IEEE 802.11b PLCP frame types

The first field of a PLCP frame is the Preamble, which contains the Sync and Start of Frame Delimiter (SFD) subfields. The preamble is transmitted at 1.0 Mbps using

Differential Binary Phase Shift Keying (DBPSK). DBPSK is a modulation method in which bits are encoded as phase shift differences between successive symbol periods.

The Sync subfield may assume two forms: long or short. The Long Sync subfield is composed of 128 "1" bits while the Short Sync subfield is composed of 56 "0" bits.

The SFD also varies with the length of the PLCP frame. For the Long PLCP frame the SFD is the sequence 1111 0011 1010 0000. For the Short PLCP frame, and to avoid confusion, the SFD takes a reverse value, 0000 0101 1100 1111.

The following field is the Header. This field contains the Signal, Service, Length and CRC subfields. Both Long and Short Signal subfields indicate the speed and transmission method of the enclosed MAC frame. However, Short PLCP frames are limited to three transmission speeds (2 Mbps, 5.5 Mbps, and 11 Mbps), meaning that they can solely be supported by networks capable of such data rates. The Service field was originally reserved for future use but it was found useful to extend the Length field, indicating the type of coding used for the packet (CCK or Packet Binary Convolution Coding (PBCC)) and indicating the use (or not) of locked clocks, which implies that the transmit frequency and symbol clock use the same oscillator.

The Length sub-field is common in both short and long PLCP frame formats and consists on the number of microseconds required to transmit the enclosed MAC frame.

The CRC field is 8-bit wide in long PLCP frames and 16-bit wide in short PLCP frames. It is calculated in the sender using the Signal, Service, and Length subfields. The receiver uses the CRC subfield to ensure that the header was received intact and was not damaged during transmission.

The last field is the PHY Service Data Unit (PSDU) that carries the MAC PDU. Figure 5-6 shows that, for Long PLCP frames, the Header field is transmitted at 1Mbps and for Short PLCP frames the Header field is transmitted at 2Mbps.

5.2 ZIGBEE and 802.15.4

The ZigBee[7] technology was developed by the ZigBee Alliance [44] and the IEEE 802.15 Task Group 4 [45] to address applications requiring long battery life and low data rates. An additional requirement, concerning the protocol stack, was that its implementation should be less complex than the existing for standard for technologies covering the same application range.

[7] The name arises from the communication method that bees use. Bees move in zigzags to share information on the position, distance and direction of the food they find.

Collaborative work between the IEEE 802.15 Task Group 4 and the ZigBee Alliance fuelled the development of the IEEE 802.15.4 standard [46]. This standard defines the PHY (Physical) and MAC (Medium Access Controller) layers for Low-Rate Wireless Personal Networks (LR-WPANs). The ZigBee protocol stack, depicted in Figure 5-7, is built on top of the IEEE 802.15.4 layers. In consequence, the ZigBee Alliance is just responsible for the Network/Security and Application layers. These provide interoperability between products of different manufacturers (among other features).

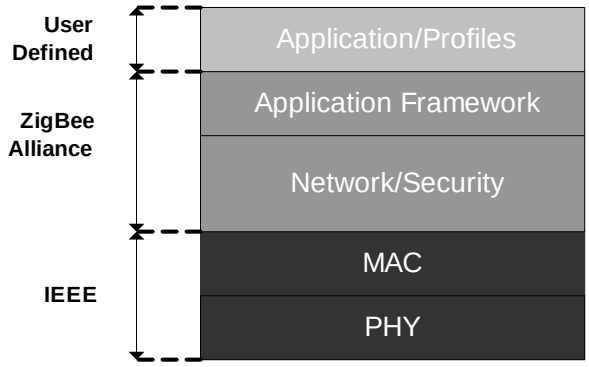


Figure 5-7 - ZigBee stack

The following subsection describes the IEEE 802.15.4 standard by focusing on the MAC and PHY layers of the protocol. Upper layers will not be discussed in this document since this work relates primarily with functionalities provided at the MAC layer level. However, if additional information (regarding higher layers) is required, it can be found in the recently released ZigBee Specification [47]. This specification describes, in detail, the Network/Security and Application Framework layers.

5.2.1 IEEE 802.15.4

As mentioned, the IEEE 802.15.4 was designed to address Low-Rate Wireless Personal Area Networks (LR-WPAN) given that it would provide short-range operation, low cost, high autonomy, low data rate and low complexity. These design goals lead the development towards the standard known today. Table 5-1 summarizes the overall IEEE 802.15.4 standard characteristics.

The IEEE 802.15.4 defines two types of participating devices in a LR-WPAN: full-function device (FFD) and reduced-function device (RFD). A FFD can operate as a Personal Area Network (PAN) coordinator, a coordinator or simply a communications device. Any FFD can talk with another FFD or with an RFD. RFDs can talk only with FFDs. RFDs are simpler and cheaper when compared to FFDs. Therefore, are limited to

become leaf nodes. Their simplicity implies that they cannot perform complex tasks and can only be used in scenarios where the amount of data for transmission is low (sensors, actuators, etc.).

Table 5-1 - Overall IEEE 802.15.4 characteristics

IEEE 802.15.4
✓ Over-the-air data rates of 250 kb/s, 40 kb/s, and 20 kb/s; ✓ Star or peer-to-peer operation; ✓ Allocated 16 bit short or 64 bit extended addresses; ✓ Allocation of Guaranteed Time Slots (GTSs); ✓ Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) channel access; ✓ Fully acknowledged protocol for transfer reliability; ✓ Low power consumption; ✓ Energy Detection (ED); ✓ Link Quality Indication (LQI); ✓ 16 channels in the 2450 MHz band, 10 channels in the 915 MHz band, and 1 channel in the 868 MHz band.

5.2.1.1 Network Topologies

The basic component of an 802.15.4 network is the device (FFD or RFD). When two or more devices communicate over the same physical channel, a WPAN is created. However, this network must include at least one FFD, operating as PAN coordinator.

The IEEE 802.15.4 standard defines two (Figure 5-8) possible topologies for LR-WPANs: star topology and peer-to-peer topology.

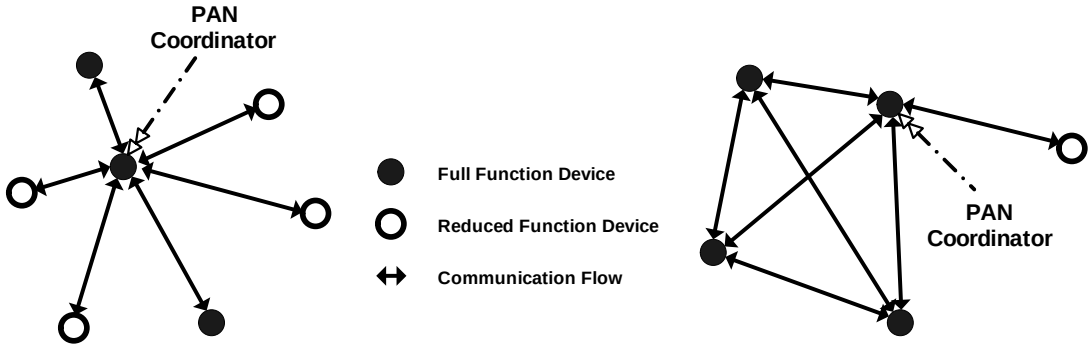


Figure 5-8 - Basic IEEE 802.15.4 network topologies

The star topology is primarily used in applications where a central node is used as PAN coordinator and where the leaf nodes act as endpoints or have some specific application. In this topology all the nodes communicate with a single central node known as PAN coordinator. The PAN coordinator can have some specific application but, simultaneously, it can be used to initiate, terminate or route data through the network.

The peer-to-peer topology also includes a PAN coordinator, but is different from the star topology by allowing communication with all devices in range (Figure 5-8). This topology provides a higher level of flexibility thus allowing the formation of more complex networks (for example mesh networks).

A peer-to-peer network can be *ad-hoc*, self-organizing and self-healing. Several functions such as message routing from any device to any other device on the network are enabled at the network layer (ZigBee Alliance).

5.2.1.2 Network Formation

Although network formation is performed at the network layer, a brief overview of the mechanism is provided here. This overview explains the formation of the star and the peer-to-peer network topologies.

Star Topology

After a FFD is activated for the first time, it may establish a network and become the PAN coordinator. Star networks operate independently allowing each PAN coordinator to choose a different PAN identifier (within its radio range). Once the PAN coordinator has chosen the PAN identifier, other devices are allowed to join its network (both FFDs and RFDs).

Peer-to-Peer Topology

The peer-to-peer topology enables devices to communicate directly within their radio range. Usually, the first device to communicate on the channel will be nominated PAN coordinator. This topology allows high flexibility on the structure of the network enabling the formation of complex networks such as, for example, the cluster-tree network.

The following subsections describe the IEEE 802.15.4 MAC layer as part of the OSI Data Link Layer (DLL) and the IEEE 802.15.4 PHY layer. The approach is mainly based on [46] and [48].

5.2.1.3 MAC Layer

Figure 5-9 presents the IEEE 802.15.4 layered structure using the International Organization for Standardization (ISO) open systems interconnect (OSI) reference model [49]. The IEEE 802 project divides the DLL into two sub-layers, the Medium Access Control (MAC) and the Logic Link Control (LLC).

This figure also demonstrates that the IEEE 802.15.4 MAC provides services to an IEEE 802.2 type I LLC through the service-specific convergence sub-layer (SSCS) or directly to a proprietary LLC sub-layer. The SSCS ensures compatibility between the MAC and different LLC sub-layers.

The IEEE 802.15.4 MAC layer supplies the following features: association and disassociation, acknowledge frame delivery, channel access, frame validation, guaranteed time slot management and beacon management.

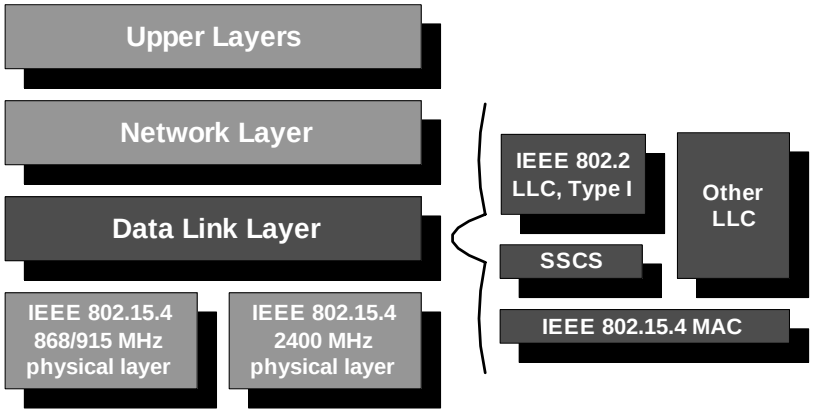


Figure 5-9 - IEEE 802.15.4 in the ISO-OSI layered network model

The MAC layer supplies two services to higher layers: the MAC data service and the MAC management service. These two services are accessed through service access points (SAPs). These SAPs are the MAC Common Part sub-layer (MCPS-SAP) and the MAC layer management entity (MLME-SAP). The MAC Common Part sub-layer has only five primitives.

Figure 5-10 illustrates their (MCPS DATA primitives) usage in the establishment of a successful data transfer between two devices.

The MAC management service has only 30 primitives, meaning that the IEEE 802.15.4 MAC is simple enough to be used in low-end applications. On the other hand, given its simplicity, it cannot perform more complex tasks (for instance synchronous voice communication as the 802.15.1 MAC).

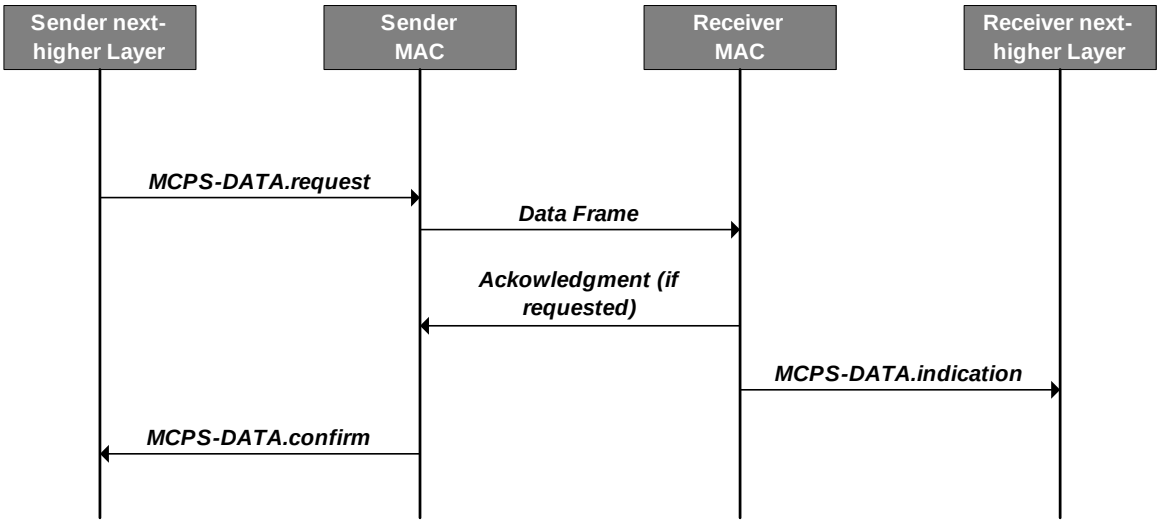


Figure 5-10 - Message sequence chart describing the MAC data service

The MAC Frame Format

The general format of a MAC Protocol Data Unit (PDU) is depicted in Table 5-2. This frame is composed of a MAC Header (MHR), a MAC payload (also known as MAC Service Data Unit (MSDU)) and a MAC footer (MFR).

Table 5-2 - General IEEE 802.15.4 MAC Protocol Data Unit

Octets: 2	1	variable	variable	2
Frame Control	Sequence Number	Addressing Fields	Data Payload	FCS
MHR			MAC Payload	MFR

The first field of a MAC frame is the Control field, which specifies the contents of the remaining fields. In this sense, it specifies the type of frame, the format of the address field and the acknowledgment control [48]. The second field is the Sequence Number that allows the matching of the Acknowledgment frames with the sent frames. Only when the Sequence Number of the Acknowledgment frame matches the Sequence Number of the sent frame is a transaction considered successful. The third field is the Address, which may vary in size from 0 to 20 bytes. The size variation is due to the nature of the frame. For instance, a data frame usually contains both source and destination addresses while an Acknowledgment frame does not contain address information. Also, the device addresses may be 8-bit or 64-bit (IEEE) long. The fifth field is the Payload, which is variable in length. This field may not exceed 127 bytes in length and the data it transports is

dependent on the frame type. Finally, the last field is the Frame Check Sequence (FCS) that allows verifying the integrity of the frame.

The IEEE 802.15.4 MAC defines four types of frames: data frame, acknowledgment frame, beacon frame and command frame.

The Command and Acknowledgment frames only are used at MAC layer level to establish peer-to-peer communication and therefore carry MAC specific data. On the other hand, Data and Beacon frames carry data from upper layers.

5.2.1.4 The Superframe structure

An IEEE 802.15.4 network can either work in beacon-enabled mode or non-beacon-enabled mode [50]. In beacon-enabled mode, a PAN coordinator broadcasts beacons periodically to synchronize the attached devices. In non-beacon-enabled mode there is no periodical beacon broadcast (from the PAN coordinator) but a soliciting device can be addressed using a unicast mechanism if required.

A superframe structure is used in beacon-enabled mode. Figure 5-11 depicts this structure, also showing that the superframe boundaries are beacon frames. The superframe structure is composed of an active part and an (optional) inactive part.

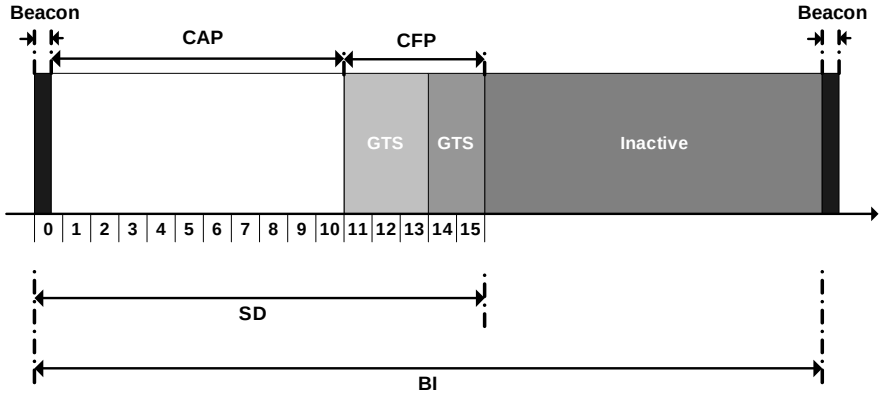


Figure 5-11 - An example of a superframe structure

The beacon interval (BI) and the superframe duration (SD) are determined either by the beacon order (BO) and the superframe order (SO). The active part is composed of *aNumSuperframeSlots* equally sized slots (being 16 the default value). Moreover, it can be further divided into a contention access period (CAP) and an optional contention free period (CFP). The former uses a slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) mechanism for channel access. All contention-based communications shall be completed before the beginning of the CFP. The latter can accommodate up to 7 Guaranteed Time Slots (GTS). GTSs refer to communication slots

where data transfers occur, without contention, between a device and the coordinator (and vice versa). Normally, the CFP is used to assign communication time slots to devices requiring dedicated bandwidth or low latency transmissions. GTSs may occupy more than one slot while guaranteeing that all transactions using GTSs complete before the next GTS or the end of the CFP.

5.2.1.5 The PHY Layer

The IEEE 802.15.4 defines two PHY layers; the 2.4 GHz and the 868/915 MHz band PHYs. Both are based on Direct Sequence Spread Spectrum (DSSS) and share the same packet structure. The most significant difference between the two PHYs is the frequency band in which they operate (see Figure 5-12 and Table 5-3).

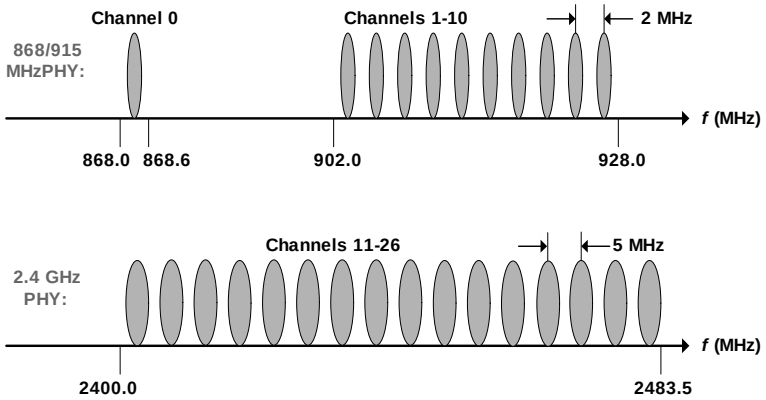


Figure 5-12 - IEEE 802.15.4 channel structure

The 2.4 GHz PHY operates in the Industrial, Scientific and Medical (ISM) band available almost worldwide. The 868/915 MHz PHY operates in the 868 MHz European band and in the 915 MHz ISM American band. Despite being available almost worldwide, the increasing use of the 2.4 GHz band may result in a large traffic congestion. The 868/915 MHz band offers an alternative PHY that, besides avoiding the excessive use of the 2.4 GHz band, it does not suffer interference from domestic equipment (for example microwave ovens).

Table 5-3 - IEEE 802.15.4 channel frequencies

Channel Number	Channel Central Frequency (MHz)
k=0	868.3
k=1, 2, …, 10	906 + 2(k-1)
k=11, 12, …, 26	2405 + 5(k-11)

Another distinguishing characteristic is the achieved transmission rate in both PHYs. The 2.4GHz band is capable of carrying data at a rate of 250 Kbit/s while the 868/915 MHz bands can achieve data rates of only 20 Kbit/s and 40 Kbit/s respectively. The higher data rate of the 2.4 GHz PHY is largely due to a higher-order modulation scheme representing multiple bits by one data symbol. Given the large application spectrum addressed by this technology, it is expected that each PHY will find its place.

Figure 5-12 and Table 5-3 show that there are 27 available channels across the three bands. However, it is unlikely that they will be used simultaneously due to the lack of regional support. An assessment of these bands must be performed prior to its usage. The MAC layer supports several low-level functions such as receiver energy detection, link quality indication and channel switching. These functions allow higher layers (network) to decide on which band to operate.

The PHY Frame Format

The PHY packet structure, also known as PHY Protocol Data Unit (PPDU), is shown in Table 5-4.

Table 5-4 - General IEEE 802.15.4 PHY Protocol Data Unit

Octets: 4	1	1		Variable
Preamble	SFD	Frame Length (7 bits)	Reserved (1 bit)	PSDU
SHR		PHR		PHY Payload

The first field is the synchronization header (SHR), which is composed of preamble and start of frame delimiter (SFD). The second field is the PHY header. The PHY header comprises two subfields: the frame length and a reserved bit. The frame length is 7-bit wide, meaning that the maximum achievable packet length is 127 bytes (also presented in the MAC subsection). However, in theory the packets could have zero-length. This will never occur, in a practical practice, due to the MAC overhead. The third field is the PHY service data unit (PSDU) that will carry the MAC protocol data unit (MPDU).

Modulation and Range

The 868/915 MHz PHY and the 2.4 GHz PHY both use Direct Sequence Spread Spectrum (DSSS). However, the 868/915 MHz PHY uses a simpler approach. Each transmitted bit is represented by a binary phase-shift keying (BPSK) modulated 15-chip maximal length sequence. The 2.4 GHz PHY employs a 16-ary quasi-orthogonal modulation technique. This means that, for each symbol period, four bits are used to select one of the possible 16 nearly orthogonal pseudo-random noise sequences (PN). Later, these successive PN

sequences (corresponding to successive data symbols) are aggregated into a chip sequence and modulated onto the carrier using Offset Quadrature Phase-Shift Keying (O-QPSK).

The specified receiver sensitivities for the 868/915 MHz PHY and the 2.4 GHz PHY are -92 dBm and -85 dBm respectively. The operational range is a function of the receiver sensitivity and transmitted power. Because the standard specifies that each device should be capable of transmitting at least 1 mW, the expected range is of ten to twenty meters. However, the actual transmitted power may be increased or decreased (within regulatory limits).

Applications requiring low latency may operate with star network topologies. The achievable range can be improved by increasing the transmit power (and also the receiver sensitivity). Applications with less demanding requirements (e.g. higher latency) may operate using the mesh network topology. This topology is known to reduce exploration costs by applying data routing algorithms to spread (or collect) data.

5.3 ULTRA-WIDE BAND

Ultra Wide-Band, often considered a recent evolution in wireless technology, experienced almost 40 years of development since its original appearance on time-domain electromagnetics [51]. The increasing interest in this technology for communication applications was triggered by modifications introduced by the Federal Communications Commission to its Part 15 rules in order to accommodate UWB transmissions. Afterwards, several proposals based on conventional modulation techniques such as Orthogonal Frequency Division Multiplexing (OFDM) and Code Division Multiple Access (CDMA) were presented. Today, two proposals are competing for its way on the market: OFDM UWB and DS-UWB. The first is based on the OFDM modulation technique and claims more spectral flexibility, which addresses the problem of lack of spectral regulatory specifications in some countries. The second is based on the CDMA modulation technique and is supported by Freescale [52], which already has a commercial implementation.

The following section will briefly describe the UWB technology from an OFDM point-of-view. Because there is not much information available for non-OFDM members, this overview will only focus on known aspects of the technology such as its physical characteristics.

5.3.1 Overview

FCC defines ultra-wideband (UWB) as any signal that occupies at least 500 MHz of bandwidth in the 7.5 GHz chunk of spectrum between 3.1 GHz and 10.6 GHz (see Figure 5-13).

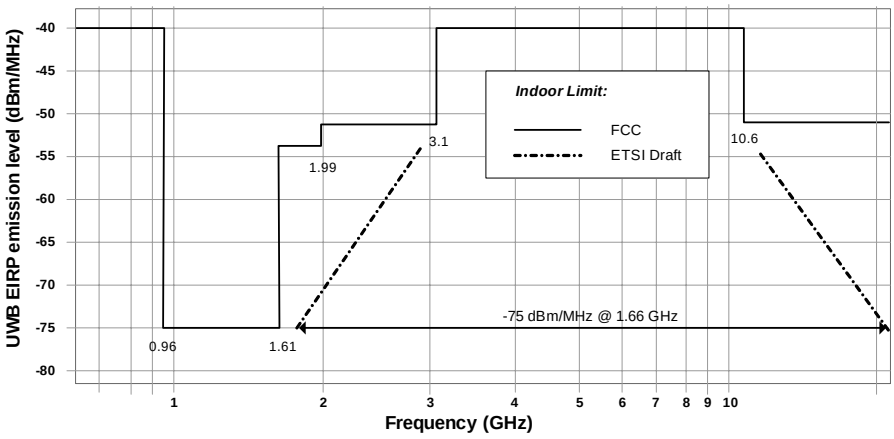


Figure 5-13 - First Report and ETSI draft spectrum mask for UWB communications in indoor scenarios

This definition has also strict restrictions concerning the maximum radiated power (radio emissions must be lower than the permitted level of incidental electronic noise in computer's switching power supplies).

The UWB technology is characterized by transmitting low-power streams of short pulses (in the order of 10-1000 picoseconds) [53] [54]. Provided that they use very high frequencies, they can be transmitted directly without the need of being previously modulated into a carrier (like in other radio systems such as Wi-Fi). The actual information is impressed onto the pulse train by varying the amplitude, spacing or duration of the individual pulses in the train.

The FCC First Report and Order [55] and the ETSI draft [56] specify the spectral masks shown in Figure 5-13. These spectral masks allow to compute the maximum permitted Effective Isotropic Radiated Power (EIRP) of 0.562 mW given the imposed power limit of 75 nW/MHz.

Since the allowed EIRP is very low, UWB radio emissions will not interfere with existing communications. This allows an enormous outcome in terms of bandwidth reuse resulting in a more efficient use of the available bandwidth. Obviously, further spectrum efficiency may be achieved by applying concepts of data routing between neighbor nodes (as in *Bluetooth scatternets* [57]).

Figure 5-14 demonstrates the UWB positioning among the remaining wireless technologies. Although its operational range is limited to around 10m, its throughput capacity is several times higher than the achieved with IEEE 802.11 technology.

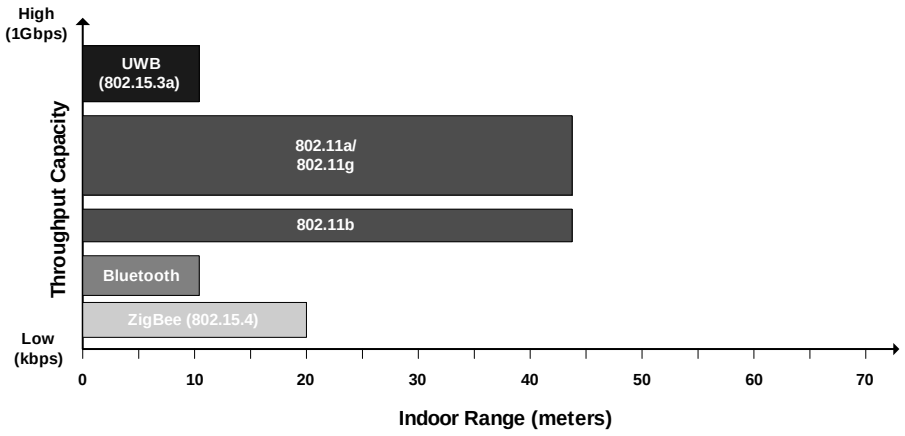


Figure 5-14 - UWB technology positioning

As noticed, the spectrum masks approved by the FCC and the more conservative ones under development for Europe by ETSI ERM TG31a ensure that the emerging UWB products will be safe and will not affect other communication technologies.

The IEEE has also established Task Group 3a (TG3a), which focuses on defining an alternative UWB PHY for the IEEE 802.15.3 standard [58]. This new Alt-PHY [59] (IEEE 802.15.3a) will work with the already specified MAC (IEEE 802.15.3). Although the adoption of a predefined MAC may reduce the overall system efficiency, it is expected that this approach will help the expedition of commercial deployment. Figure 5-15 shows the UWB protocol stack as defined by the Multiband OFDM Alliance.

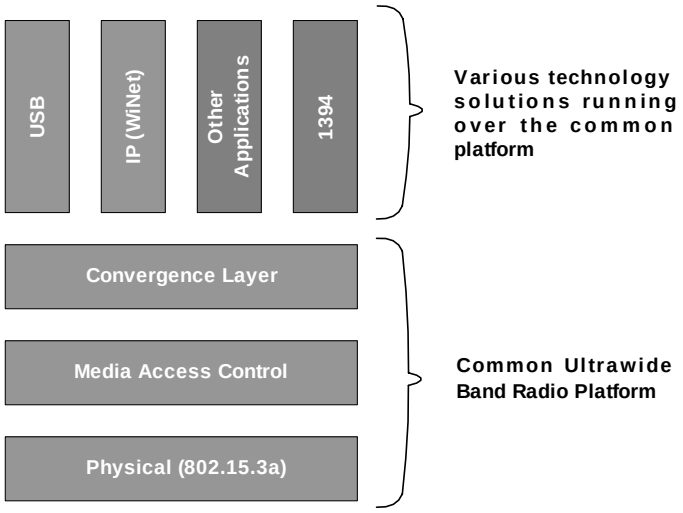


Figure 5-15 - UWB protocol layers and application

As it can be seen, the physical layer will be based on the IEEE 802.15.3a PHY proposal and the MAC layer on the IEEE 802.15.3 MAC. Several other technological solutions may

run on the top of the UWB protocol stack. Examples such as WiNet, UWB USB and UWB 1392 are already being explored.

The successful effort in branding wireless technologies demonstrated by the Wi-Fi Alliance [60] motivated the creation of a similar association called WiMedia [61]. This alliance was created in 2002 in order to build a brand image, and establish test and interoperability compliance procedures for the IEEE 802.15.3 standard.

Concluding, the Ultra Wide-Band technology promises a revolution in the WPANs by providing extended data rates up to 480 Mb/s and a low level of radiated power (maximum of 0.562 mW). Power consumption will be reduced and the interference with other wireless technologies (operating in the same spectrum) minimized. Despite of these advantages, UWB was not considered due to its lack of commercial availability, at the time of this writing.

5.4 Conclusion

This chapter described several technologies alternative to *Bluetooth*. The approach was to describe firstly the wireless technologies with more field implementations and lastly, the more promising technologies.

Wi-Fi is a successful communication technology but presents a higher cost and power consumption than *Bluetooth*. However, it provides higher data rates and extended operational range. ZigBee only recently reached commercial implementation and therefore, is not yet sufficiently cost-effective to be considered as a viable solution. Likewise, UWB is being developed and promises several features that may reinforce the use of wireless communications (for example, power consumption and throughput). Nevertheless, at this point, *Bluetooth* still seems a better option for the wireless transmission of MIDI.

CHAPTER 6

WIRELESS MIDI - BLUETOOTH BASED SOLUTIONS

6.1 Introduction

Several solutions could be sought to transmit MIDI in wireless channels using *Bluetooth*. This work presents three of them: the Serial Port Profile, the use of an Asynchronous Connectionless Link (ACL) and the use of an ACL link altogether with a command aggregation algorithm. These approaches were considered on a top-down perspective, first evaluating solutions already implemented (there are several vendors selling commercial RS232-*Bluetooth* modules using the SPP) and after developing our own solutions by identifying limitations in the available ones.

The Serial Port Profile approach was considered because it enables a direct mapping of standard wired MIDI connections over wireless *Bluetooth* links, thus making the connection procedure easy to the user. Another important characteristic is that this profile

is one of the most widespread and has large support by the *Bluetooth* industry. Despite these facts, there are no documented studies on its timeliness and, consequently, no guarantees of fulfilling the MIDI timing requirements.

The second approach uses Asynchronous Connectionless Links (ACL) to carry MIDI data flows. This approach is less complex than the SPP because it just requires half of the *Bluetooth* stack. It was proposed mainly to evaluate the influence of the stack in the *Bluetooth* timeliness. In addition, it was used to identify the timing restrictions of a minimum wireless MIDI (WiMIDI) implementation.

The ACL with command aggregation approach was developed to overcome the limitations found in the ACL based solution, which jeopardized the transmission of MIDI chords. Therefore, a command aggregation algorithm is proposed and analyzed in detail.

Although several solutions are proposed, they all share a common architecture like the one proposed in chapter 3. The following sections describe the envisaged operation of the system and the architecture of the test-bed used to assess the performance of the system. Further on, each approach is studied in detail and the experimental results obtained are discussed.

6.2 Standard Operation

The proposed WiMIDI architecture maps directly common MIDI connections on wireless links, ensuring a similar user experience as the one obtained in normal wired connections. In this sense, WiMIDI links are unidirectional and replace directly physical MIDI cables. WiMIDI devices are configurable and emulate a MIDI connection after proper setup. By configuring a WiMIDI device to act as a sender and another as a receiver, a connection can be established.

Musical performances frequently require several instruments to be played simultaneously. Therefore, it is necessary to define individual identifiers, here called keys (shared by both WiMIDI devices, the sender and the receiver), for each WiMIDI connection. In the setup phase keys are used to match WiMIDI pairs of devices correctly.

In chapter 3, Figure 3-6 illustrated two different connections using WiMIDI devices for cable replacement. As explained, the WiMIDI sender device (for example WiMIDI IN01) is connected to a standard MIDI output, whereas the receiving device (for example WiMIDI OUT01) is connected to an input. All MIDI traffic arriving from the output of the MIDI device will be wirelessly transmitted into the input of the other device. Notwithstanding, the existence of multiple WiMIDI connections in a reduced geographical space, they are independent concerning MIDI flows from different instruments. The pairing procedure ensures a unique ID for each connection, which means that if an ID is in

use, no other pairing procedures are allowed to use it. In other words, when a user selects two WiMIDI devices and tries to set a Wireless MIDI link between them, an identification (ID) for the connection must be chosen. If the chosen ID is already in use, the WiMIDI devices will both return an error message indicating that a different ID must be chosen.

In [62] an upper bound of 10 simultaneous piconets using one-slot packets, operating on the same coverage area and performing at the maximum throughput, is defined. This means that more piconets can coexist in the same geographical space, but operating with lower throughputs. Since the required throughput of the application is a function of the human response time, it will be presumably low when compared to the throughput the communication link has available. This means that probably, the proposed architecture will be able to support a larger number of simultaneous piconets. This reasoning can only be validated though the characterization of the MIDI data flows in a typical music performance, which is beyond the scope of this work. Particularly, in this implementation, only 10 different IDs are allowed to be used, since the user interface of a WiMIDI unit (7-segment display) can only represent numbers ranging from 0 to 9. These numbers are then used as a part of the *Bluetooth* PIN, thus facilitating the identification of the WiMIDI networks.

The following section describes the test-bed used to assess the performance of the proposed approaches.

6.3 Test-Bed Architecture

Aiming to evaluate the time response of the proposed approaches, a testing system was developed. This delay measurement system (DMS) was built with the purpose of measuring delays and their variation. As noticed in previous sections, MIDI has tight requirements concerning command/chord delay and jitter. Therefore, the performance of the approaches has to be expressed as function of these parameters.

The measurement system also addresses other requirements such as the ability to measure the delay between MIDI bytes or MIDI commands, always in a non-destructive manner. In other words, the measurement system only “watches” the MIDI flow at both communication ends and computes the correspondent delays without affecting the MIDI flow.

Figure 6-1 shows an illustrative diagram of the test-bed architecture. Two personal computers assume the role of MIDI source and MIDI sink. This way, the MIDI data flow is fed to the DMS using the PC on the left. The system records the instants in which MIDI bytes (or commands) are sent or received by the source or sink, respectively. In addition, this information is sent to a separate PC using a fast serial connection. On a subsequent phase, the recorded data is processed offline and used to compute MIDI delays.

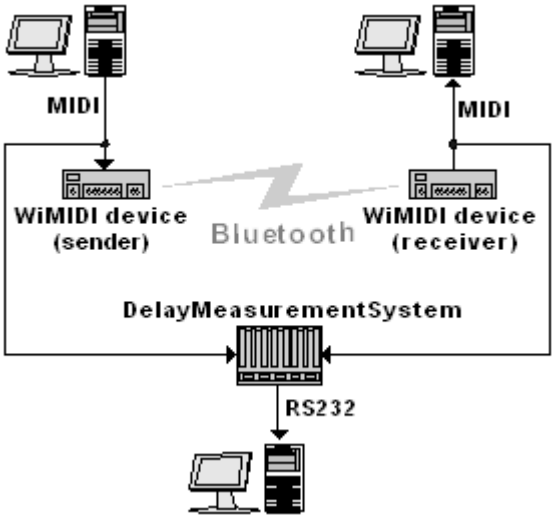


Figure 6-1 - Delay measurement system (Test-Bed)

Further on, it will be seen that the hardware composing each WiMIDI device [63] is highly dependent on the discussed approach. Therefore, the hardware analysis will be left to be presented there. However, the measurement system remains unmodified throughout all approaches. For this reason, it is discussed in the remaining of this section

The Delay Measurement System (DMS) consists of two Cygnal C8051F040 micro-controller evaluation boards running at 22MHz. The same external oscillator running at 32KHz drives both boards. This particularity has been included to ensure an accurate synchronism between both boards. The delay measurement occurs in terms of ticks, each one lasting approximately 1/32.768 milliseconds.

Each Cygnal development board is connected to one of the MIDI ends by means of a MIDI-Serial electrical interface. This interface provides electrical conversion from MIDI levels into RS232 levels. Each development board can be programmed to register single byte or full command instants.

The Delay Measurement System operates by sniffing the MIDI physical connection in search for new bytes or commands. After a successful detection, an RS232 message, containing the detected data and the instant in which it has been sent (or received), is sent to the logging PC. This PC stores all received messages in two separate files, one with the transmission instants and the other with the receiving instants. These instants are later used to compute the corresponding delays.

The following section describes the proposed approaches for using *Bluetooth* as the carrier technology for MIDI command streams. Several measurements were made and will be thoroughly discussed.

6.4 Solutions

6.4.1 Serial Port Profile

The first considered approach was to use the Serial Port Profile (SPP) in order to take advantage of its serial port emulation feature [64]. The MIDI protocol highly resembles with a particular configuration of the RS232 protocol and therefore, this solution seemed a good starting point to communicate MIDI data. Another reason is the fact that it is well supported by the *Bluetooth* industry. In addition, it privileges a top-down approach to existing solutions.

6.4.1.1 Hardware

The developed WiMIDI nodes (Figure 6-2) are based on a Cambridge Silicon Radio BlueCore2-External device that uses the Serial Port Profile implementation from Mezoe, which is available with the BlueLAB SDK [31]. This device is compliant with the *Bluetooth* specification v1.1.

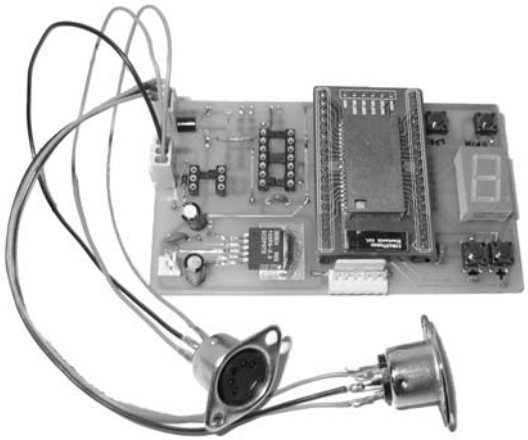


Figure 6-2 - SPP WiMIDI prototype

The BlueCore2-External is a complete single-chip *Bluetooth* solution with onchip microcontroller, and a 4Mbit external Flash memory interface for general-purpose usage and development. This configuration allows a high degree of flexibility in terms of the solution deployment. The application can run on the top of the *Bluetooth* stack within the device or it can run on a separate microcontroller interfacing with the *Bluetooth* device. This device can be easily upgraded using specific software provided by CSR, thus making it flexible enough to implement different solutions.

6.4.1.2 Performance Evaluation

The performance evaluation of the *Bluetooth* SPP was made using the described test bed and hardware. For this purpose the MIDI data source was configured with the following settings:

- Random MIDI message lengths (uniformly distributed) between 1 and 6 bytes.
- Random MIDI message intervals (uniformly distributed) between 0 and 200 milliseconds.

The WiMIDI sender unit was configured to transmit a byte, in the emulated *Bluetooth* serial connection, whenever it is received from the MIDI link.

Two sets of experiments were conducted using these settings, one in which the *piconet* master operates as MIDI transmitter and the other in which it operates as MIDI receiver. Twelve individual trials were conducted for each experiment (*piconet* master or *piconet* slave transmitting). Each trial consists of transmitting 8192 MIDI bytes through an emulated *Bluetooth* serial connection and measuring the individual byte delay. This number of bytes was chosen provided that it leads (in average) to experiments lasting 4 minutes and 20 seconds, being a near realistic approach to a song duration.

The chosen environment is the Electronics and Telecommunications Department of the University of Aveiro. This department has an 802.11b wireless infrastructure network through which students access the World Wide Web. The environment was chosen considering an almost "worst case" scenario, in which several noise sources (Wi-Fi, *Bluetooth*, etc.) would exist to interfere with the *Bluetooth* links.

Figure 6-3 shows the resultant byte delays of a particular test for the *piconet* slave transmitting.

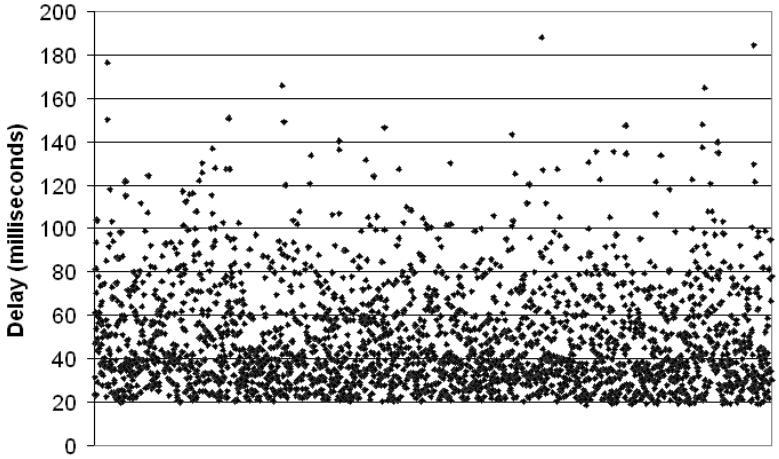


Figure 6-3 - SPP Slave-transmitting delays.

As it can be seen, there are a large number of MIDI bytes suffering delays higher than 100 milliseconds. These delays range from approximately 20 milliseconds to almost 200 milliseconds and lead to a delay variation that can reach 150 milliseconds.

Using the data that led to Figure 6-4, it is possible to verify that the delay variation is limited to 20 milliseconds for approximately 63% of the transmitted MIDI bytes. Additionally, it can also be determined that delay variations higher than 100 milliseconds occur for only 0.5% of the transmitted bytes.

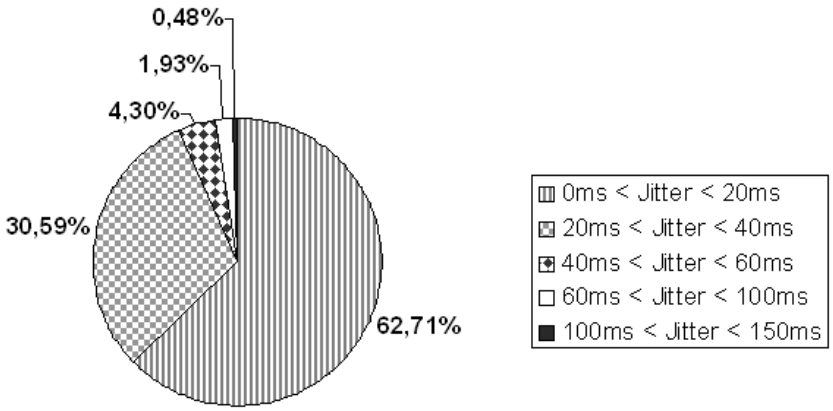


Figure 6-4 - SPP Slave-transmitting jitter incidence rate.

Figure 6-5 shows now the measured byte delays when the *piconet* master is transmitting the MIDI flow. In this experiment, byte delays did not exceed the 100-millisecond threshold and the delay variation has decreased significantly.

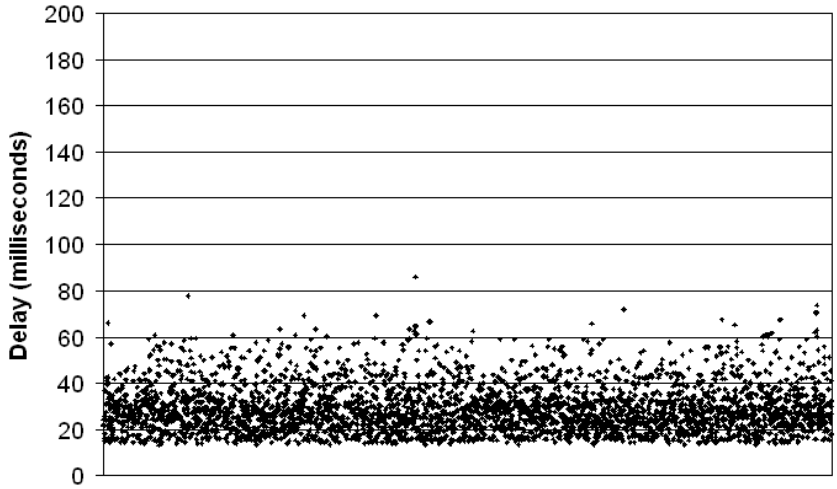


Figure 6-5 - SPP Master-transmitting delays.

Figure 6-6 documents an improvement in delay variation. When the master is transmitting a MIDI flow over a *Bluetooth* emulated serial port, MIDI bytes suffer a delay lower than 20 milliseconds in almost 96% of the situations. This is a remarkable improvement when compared with the previous scenario, were the same delay variation occurred only for 63% of the transmitted bytes.

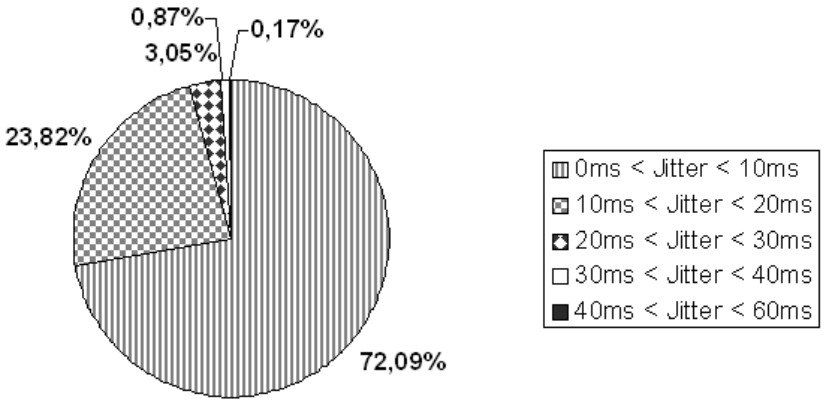


Figure 6-6 - SPP Master-transmitting jitter incidence rate.

After the completion of the two sets of experiments, several statistical variables were considered, namely the Average, Maximum and Minimum Delays as well as the Delay Variation (Jitter).

This information can be used to evaluate the performance of the approach. In this sense, best-case and worst-case delays have been computed, for each experiment, as well as the average delay. Best-case and worst-case scenarios are important to identify the operational boundaries of the approach. In addition, since the human hear can perceive small delay variations, the jitter was also computed.

Figure 6-7 indicates the statistical data obtained from the first set of experiments (*piconet* slave transmitting). With this figure it becomes visible that very high delays (about 510 milliseconds) can occur. This means that, for a given MIDI byte, the experienced delay was almost half a second. If a musician senses that a note has not yet been played (after pressing a key of a keyboard for example), he/she may be lead to think that the key had not been pressed sufficiently and will try again. Considering that the first event was transmitted with a large delay (say, half a second) and the second was transmitted with a short delay, they can become effective at the receiver almost simultaneously. Therefore, the first note will be played out of time and an additional note will sound even worse.

In Figure 6-7, it can also be observed that the average delay is lower than 50 milliseconds and that its average variation never reaches the 20-millisecond boundary. Nevertheless, these measurements merely indicate that, in average, delays are confined to the 50-millisecond threshold. However, they can be as higher as 500 milliseconds, which is unacceptable for the application in analysis.

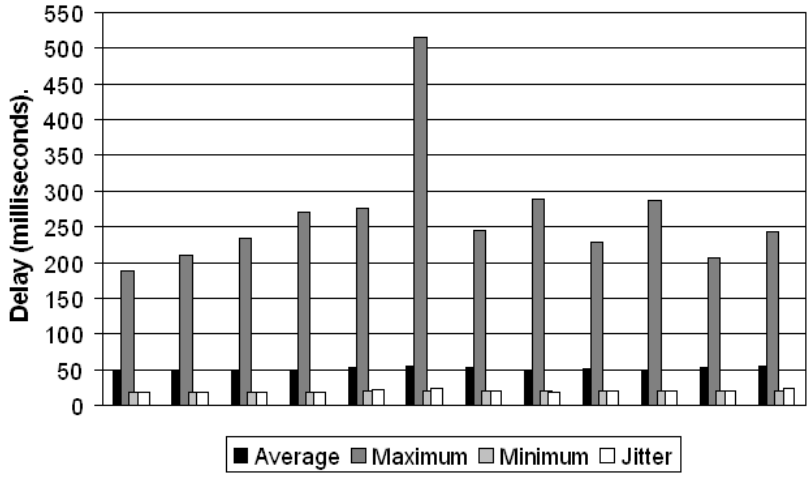


Figure 6-7 - SPP Slave-transmitting statistics.

As in the former case, measurements were used to compute the minimum and maximum delays as well as the average delay when the *piconet* master performs the role of transmitter. In addition, the average delay variation was computed. The statistical values obtained can be observed in Figure 6-8.

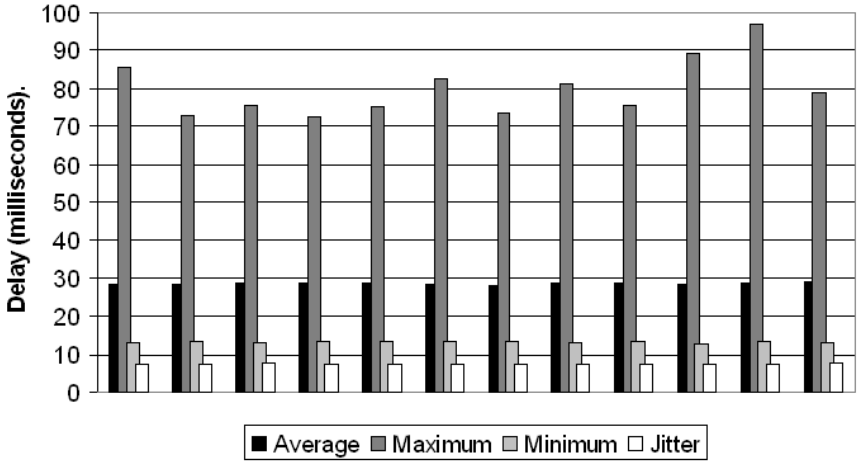


Figure 6-8 - SPP Master-transmitting statistics.

As it can be seen, all statistical values have decreased when compared to those in Figure 6-7. The maximum delay never reaches the 100-millisecond threshold and the delay is, in average, 30 milliseconds long. Furthermore, and because delays not only have decreased but have also become more regular, the average delay variation has been reduced to 10 milliseconds.

Recalling the Bluetooth standard, the *piconet* master must poll a slave before this one is allowed to transmit. On the other hand, a *piconet* master may transmit almost anytime (in even-numbered slots). It could then be presumed that commands transmitted by the *piconet* slave would experience higher delays than commands from the master. However, and considering that the *piconet* master in this *Bluetooth* stack implementation polls the slave at the maximum frequency (every two time slots for single slot packets), this explanation does not seem credible since the observed delays are much higher than 1.25 milliseconds. Instead, if it had been assumed that, for this *Bluetooth* stack implementation, the *piconet* master had a poll interval larger than two time slots, the *piconet* slave would have experienced much larger delays when trying to transmit a MIDI command.

Although this could explain why the master experiences smaller delays, it does not explain the large delays observed in general. These may relate to the *Bluetooth* flow control (already discussed in the *Bluetooth* overview) and with the specific SPP implementation. Because the SPP and the flow control mechanism are implemented through several stack layers, they may introduce large delays. However, these assumptions can only be validated using alternative SPP implementations, which is currently out of the scope of this work.

Despite MIDI transmissions over the *Bluetooth* serial link are bounded by a 100-millisecond delay and a 10-millisecond delay variation (when the *piconet* master is transmitting) they seem yet distant from the requirements of the application due to excessive delay.

6.4.2 ACL Connection

The Serial Port Profile (SPP) approach showed to be insufficiently predictable to transmit MIDI command streams. Not only the delay is excessively high but it also occurs with a large variation. To outcome the limitations of the SPP approach, a second solution is here introduced. This approach is based on the principle that layers introduce delays and contribute to higher jitter due to processor load variations. In this sense, this approach cuts down as much layers as possible (see Figure 6-9) using a pre-assembled *Bluetooth* device. Therefore, the solution works on the top of the HCI layer to establish and maintain ACL connections and to allow data communication between two *Bluetooth* devices.

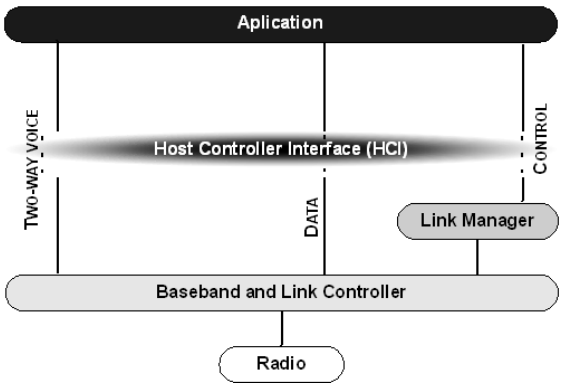


Figure 6-9 - Reduced Bluetooth stack.

This simplified approach eliminates delays introduced by the L2CAP and RFCOMM layers, thus reducing the end-to-end delay. Because the processor load will be substantially reduced, it is expected that the delay variation will also decrease.

6.4.2.1 Hardware

In this approach, WiMIDI nodes are assembled using the most common arrangement: one host and one host controller for each WiMIDI node (Figure 6-10). The reasoning underpinning this arrangement is that the firmware included in the Bluetooth module, for supporting both stack and application in the same device, includes a complete Bluetooth stack, which cannot be handled at the HCI level. The host is a Cygnal C8051F040 micro-controller evaluation board and the host controller is a CSR BlueCore2-External device flashed with the HCI *Bluetooth* stack version 1.1. The evaluation board has two serial ports; one is connected to the MIDI-RS232 electrical interface, and the other one to the host controller. The MIDI-RS232 electrical interface converts MIDI levels into RS232 levels or RS232 levels into MIDI levels, according to the MIDI flow direction.

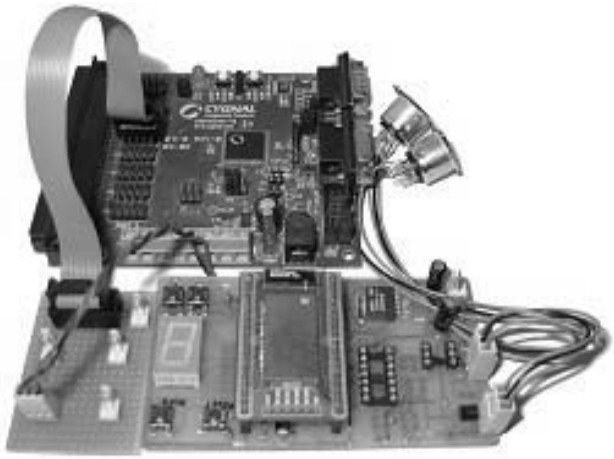


Figure 6-10 - ACL WiMIDI prototype

Each WiMIDI node is connected to one of the communication ends through the MIDI-RS232 interface. After the initial configuration, in which it is required to choose an unused ID for the network and the role of each WiMIDI device (sender or receiver), an ACL link is created between them. This link carries MIDI data (in the configured direction) and control information from one end to the other (in both directions).

The architecture of this approach has the disadvantage of requiring additional hardware (Cygnal C8051F040 based board) when compared with the SPP approach. In the previous approach there was no host because the *Bluetooth* stack and the application both resided in the same device. Nevertheless, this approach is more flexible, since it allows specifying some parameters of the connection such as the maximum number of slots a packet can occupy. For the following experiments, only one-slot packets were used.

6.4.2.2 Performance Evaluation

The performance evaluation of this approach was carried out using the Delay Measurement System, already discussed, altogether with two WiMIDI nodes (now incorporating one Cygnal C8051F040 board each). For this purpose the MIDI data source was configured using the following settings:

- Random MIDI message lengths (uniformly distributed) between 1 and 6 bytes.
- Random MIDI message intervals (uniformly distributed) between 0 and 200 milliseconds.

The WiMIDI sender unit was configured to transmit the received MIDI commands, in the ACL link, only when the last transmitted ACL packet is acknowledged.

As in the former approach, two sets of experiments were conducted, one in which the *piconet* master transmits the data stream and the other in which it listens to it. Both sets were carried on in the Electronics and Telecommunications Department and each one consists on twelve individual trials. A single trial comprises the transmission of 8192 MIDI commands with variable length (1 to 6 bytes) and the measurement of the delay each command suffers.

Figure 6-11 shows the command delays measured when a *piconet* slave is transmitting the MIDI stream (for a particular trial). As it can be seen, delays are mostly distributed around specific values that are approximately 20 milliseconds apart from each other.

This separation may be a consequence of the fact that, a slave can only transmit after being polled by the master. However, since this approach uses HCI commands to manage the ACL links, this behavior can only be explained by additional delays introduced in the lower layers of the *Bluetooth* stack. Therefore, when a slave tries to transmit an ACL

packet using an HCI command, the packet will be queued for transmission. Then, it will only be transmitted after the lower layers gain transmission access to the medium. Since the approach uses the shortest possible *Bluetooth* packets (DM1/DH1), it would be expected that the delays would be of one (0.625 milliseconds) or two timeslots (1.25 milliseconds), in the worse case. However, and since the lower layers introduce additional delays that seem to be deterministic, MIDI commands will face a delay around multiples of 20 milliseconds starting from 5 milliseconds.

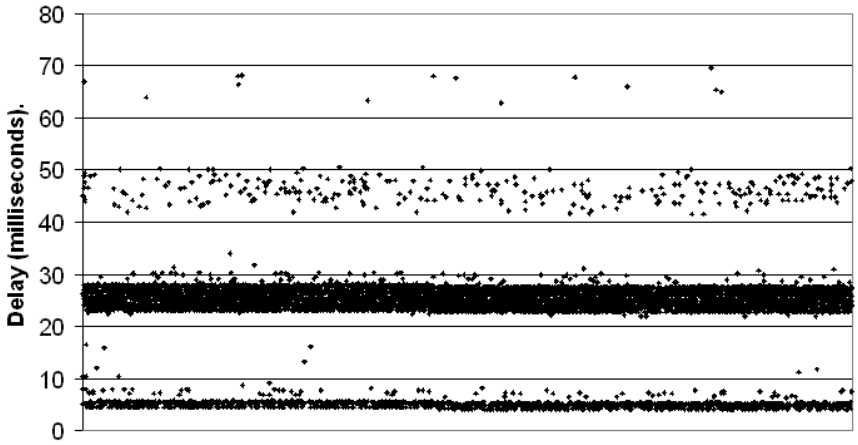


Figure 6-11 - ACL Slave-transmitting delays.

Figure 6-12 shows the delay variation incidence rate that MIDI commands suffer when a *piconet* slave has the transmitting role. It is noticeable that almost 80% of the commands suffer a delay variation lower than 10 milliseconds. Despite that fact, the maximum delay variation can reach the 50-millisecond threshold, although this situation occurs with negligible frequency.

Notwithstanding these encouraging results, there are still strong limitations for the envisaged application. For example, with the slave transmitting the MIDI stream, there are a large number of MIDI commands (20%) that experience delay variations higher than 10 milliseconds. This, of course, does not cope with the timing requirements of MIDI. In this sense, better solutions have to be investigated in order to provide an overall delay variation bellow the 4 millisecond audible threshold. This means that, in a scenario were chords are present (and in most cases they are), the communication technology must ensure that the overall delay variation (regarding all commands that constitute the chord) is lower than 4 milliseconds.

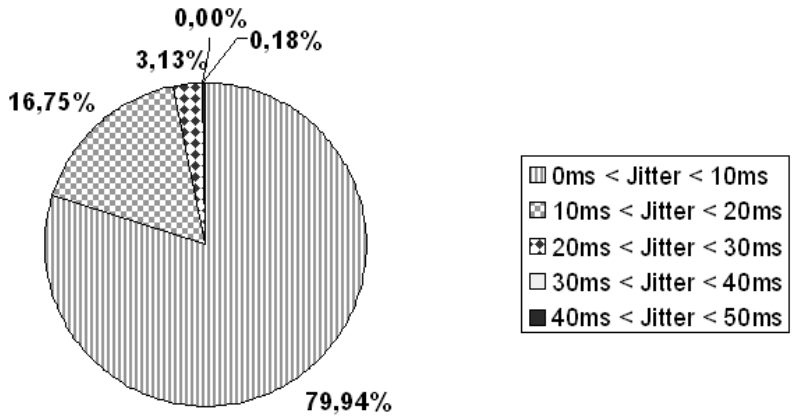


Figure 6-12 - ACL Slave-transmitting jitter incidence rate.

Figure 6-13 depicts the command delay distribution of one particular experiment performed for a master transmitting the MIDI data stream. As in the former scenario (slave transmitting), delays seem to be distributed around specified values. Nevertheless, it seems that the *Bluetooth* link introduces delays multiple of 1 millisecond (starting from approximately 16 milliseconds). Furthermore, some commands will face a higher delay that can reach the 30-millisecond threshold.

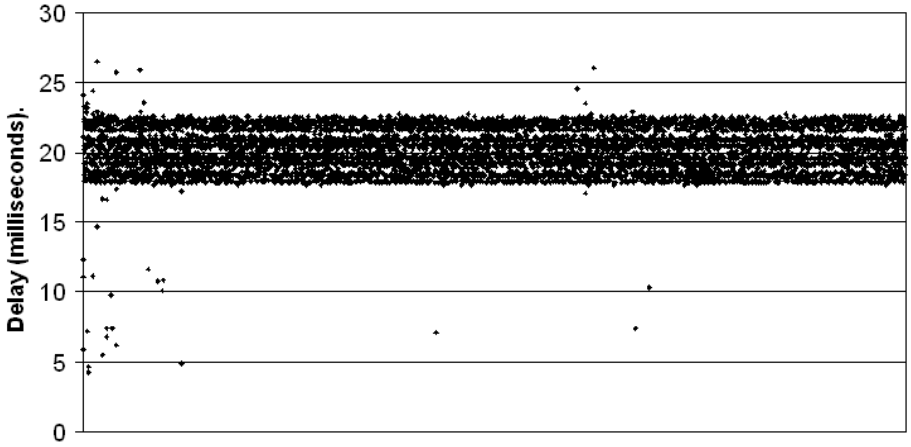


Figure 6-13 - ACL Master-transmitting delays.

Figure 6-14 shows the delay variation incidence rate that a MIDI command will face when the transmitting device is configured as *piconet* master. As it can be seen, almost 99.6% of the delay variation is lower than 3 milliseconds, which is an appreciable improvement when compared to the slave-transmitting scenario. Almost 50% of the delay variation is lower than the 1-millisecond threshold and almost 75% are within the 2-millisecond boundary.

MIDI chords usually accommodate three notes, each one produced by an individual MIDI command. Moreover, the overall delay variation of a MIDI command will be a function of the individual command delay variations. Therefore, provided that an individual MIDI command faces a delay variation of approximately 3 milliseconds, the overall delay variation will become considerably larger than the 4-millisecond threshold. In fact, in this scenario (with a three-note MIDI chord), the overall delay variation is of approximately 9 milliseconds. Clearly, it is higher than the maximum allowed delay variation.

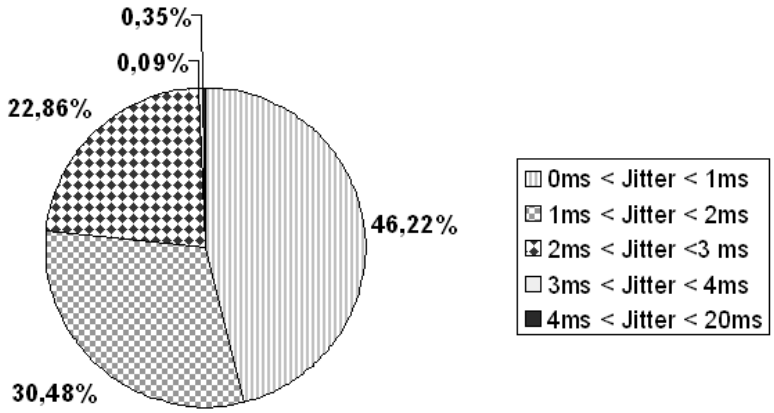


Figure 6-14 - ACL Master-transmitting jitter incidence rate.

In order to understand and evaluate the results obtained from the experiments (when a slave was transmitting), several statistical variables were calculated, namely the average, maximum and minimum delays and the average delay variation. As represented in Figure 6-15, the average delay is always lower than 25 milliseconds and the average delay variation is lower than 10 milliseconds. These results indicate that this approach does not yet fulfill the timing requirements of the application in analysis.

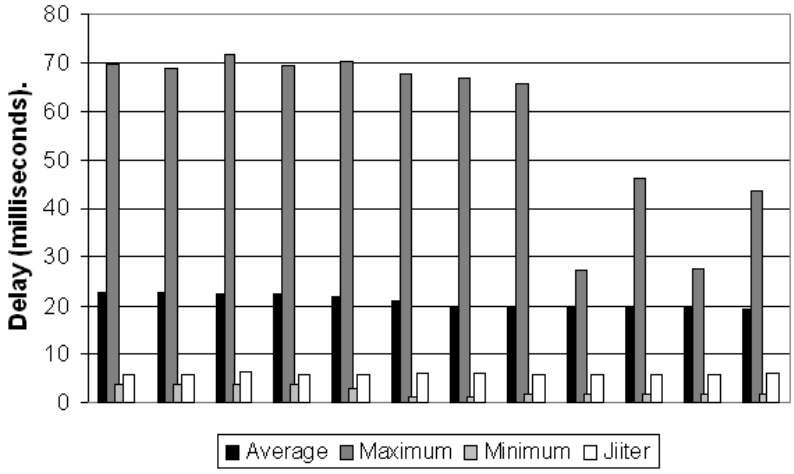


Figure 6-15 - ACL Slave-transmitting statistics.

As in the previous scenario, several statistical variables (shown in Figure 6-16) were also calculated using data from the master-transmitting experiments. It can be observed that the average delay is confined to the 20 milliseconds threshold and that the average delay variation decreased significantly (1 millisecond) when compared to the scenario where the *piconet* slave was transmitting the MIDI data stream.

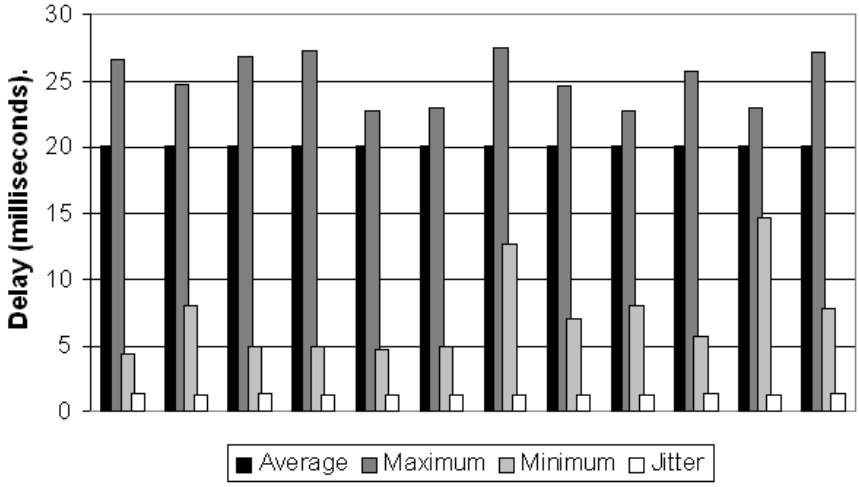


Figure 6-16 - ACL Master-transmitting statistics.

These results indicate that this approach (master-transmitting scenario) seems feasible to carry MIDI commands with single notes. In average, MIDI commands will face a 20-millisecond delay that will change within an average boundary of 1 millisecond (up and down). This will ensure that, for example, a MIDI Note-On command will become effective after approximately 20 milliseconds ±1 millisecond (in average).

As described, single note MIDI commands will face an average reduced delay variation. Because there can be higher delay variations, this approach seems only feasible to transmit single note MIDI commands. When considering chords, the overall delay variation is approximately the sum of the individual command delay variations, which may become (even in average) larger than the specified 4-millisecond threshold. In this sense, the following section proposes an alternative approach that will improve the performance of the WiMIDI system, when transmitting chords.

6.4.3 MIDI Command Aggregation

The mapping of MIDI data streams in ACL connections has shown to be feasible when the transmitting WiMIDI unit is the *piconet* master. However, as measured, it only operates within the application requirements for single note commands. The extension of this behavior to MIDI chords can be accomplished using a command aggregation mechanism. This aggregation mechanism is based on the evidence that packets sent at the HCI level

experience similar delays for different lengths. The reasoning behind this fact is that the lower-level stack introduces, in the process of breaking HCI ACL packets into *Baseband* packets, significant delays when compared to the transmission delay. Therefore, the transmission time can be neglected (even if a single HCI payload is broken into several *Baseband* packets) when compared to the maximum delay variation the application can support. If this premise holds and assuming constant length chords, it is possible to design an approach that will ensure minimum delay variations in both scenarios: single notes and chords.

The flowchart depicted in Figure 6-17 illustrates the command aggregation algorithm.

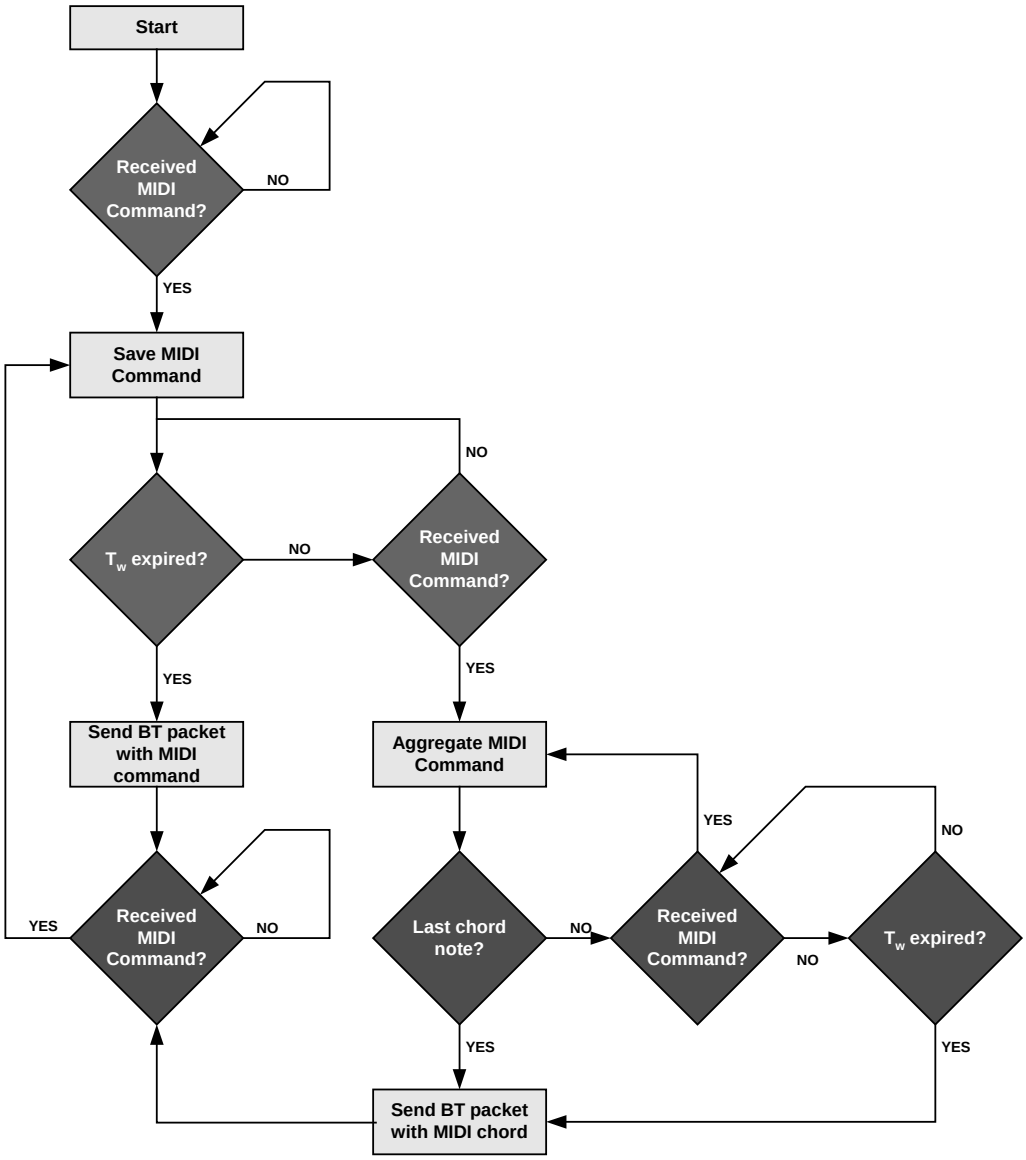


Figure 6-17 - Command aggregation algorithm

This algorithm operates by joining together the commands of a MIDI chord (three *Note On* commands for example) in a *Bluetooth* HCI ACL data packet. In other words, MIDI commands belonging to a chord will be grouped together and transmitted in the same HCI ACL packet.

When a command is received at the transmitting WiMIDI unit it can be either a single note MIDI command or it can belong to a MIDI chord. Assuming that the maximum delay between two consecutive *Note On* (or *Note Off*) commands is T_{bc} then the WiMIDI unit must wait T_{bc} seconds to check if another note command is received. This time window will be referred in the remaining text as T_w. If, within the time window T_w, no other commands are received, the system can determine that it is a single note command. Otherwise, the command belongs to a MIDI chord. In this case the WiMIDI unit must receive the remaining note commands and aggregate them to ensure the desired timely performance.

Figure 6-18 illustrates the transmission of MIDI chords using the ACL command aggregation algorithm. As noticed, T_w is the time window (that must be equal or larger than the maximum time between commands – T_{bc} – in a MIDI chord), in which commands must arrive in order to be considered part of a MIDI chord. The delay between the transmission instant of a *Bluetooth* HCI ACL packet (containing one, or more, MIDI commands) and the instant when it becomes effective is denoted by T_d. T_{scd} and T_{chd} denote the delays between the arrival of a command, or a MIDI chord, at the sender WiMIDI node and its arrival at the receiver WiMIDI node. Notice that, for visualization purposes, the command duration is expanded regarding the time between commands, e.g., they occupy more time in the figure than they really do in a practical perspective.

From Figure 6-18 it is possible to deduce an equation that accounts for the delays experienced by a single note command:

$$T_{scd} = T_c + T_d + T_w$$

In a similar manner, it is also possible to deduce an equation for the delay that a MIDI chord experiences:

$$T_{chd} = nT_c + T_d$$

where *n* is the number of commands in the chord.

Considering that both T_w and T_c are negligible when compared to the *Bluetooth* transmission delay (and they usually are) then:

$$T_{dsc} \simeq T_{chd}$$

The delay experienced by single note MIDI commands will be approximately equal to the delay experienced by MIDI chords. Therefore, the delay variation in these two scenarios will be minimized as desired.

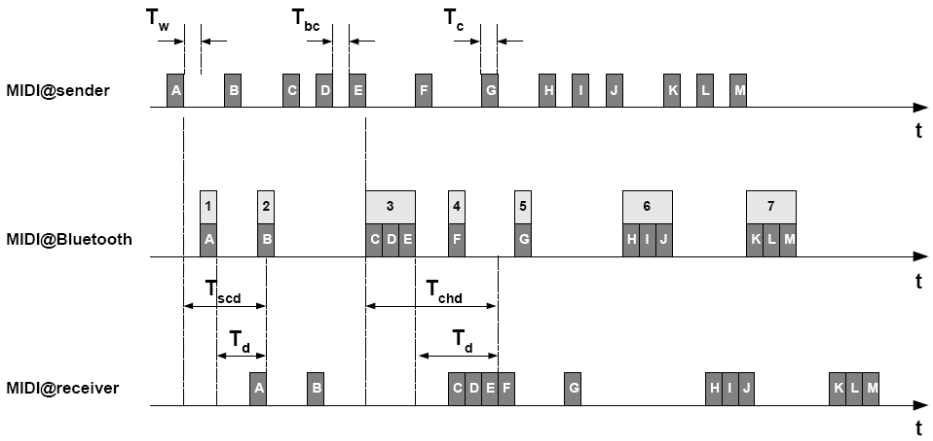


Figure 6-18 - MIDI transmission timings

6.5 Discussion

To transmit MIDI command streams using the *Bluetooth* technology three proposals were presented. The first approach was to use the direct mapping of the MIDI protocol on the Serial Port Profile, given its dissemination on the consumer market. This approach showed to be quite far from the requirements set by the application. Particularly, it was observed that delays introduced by the transmission (when the slave was transmitting) could reach half a second. However, improvements were observed when the transmitting role was assigned to the *piconet* master. In this scenario, not only the delay has been reduced, but also its variation (jitter). Nevertheless, the improvements were not sufficiently expressive to cope with the requirements of the application.

The second approach tried to verify if the delays were introduced by the communication stack or by the medium access contention (in the presence of noise in the ISM band). From the measurements, it has become evident that the *Bluetooth* stack introduces very large delays in the transmission process. For this reason, the ACL approach uses a minimum number of layers by operating directly on the top of the HCI interface. This approach showed to be feasible when the *Bluetooth* master transmits the MIDI data stream, but simply for single note commands.

The last approach, command aggregation in *Bluetooth* HCI packets, is a simple mechanism that seems to be a good option for using *Bluetooth* to transmit not only single note commands, but also MIDI chords. This approach aggregates the traffic at the transmitter

and does not require any particular synchronization between the sender and the receiver WiMIDI nodes, as it would be needed if a time-stamping mechanism had been used instead.

CHAPTER 7

CONCLUSION

The Musical Industry has grown largely in the last decade and today it constitutes an industrial niche that cannot be disregarded. Musical performances have also grown in complexity and increasingly require creative freedom. Since MIDI is the *de facto* communication protocol in the musical industry, this dissertation described the general characteristics of the MIDI protocol and provided an in depth overview of the command structure. Particular requirements, concerning a possible mapping on wireless communication technologies, were also discussed. In fact, any solution feasible for interconnecting musical instruments must offer support for mobility and flexibility. Today, this can only be accomplished using wireless communication technologies. Consequently, this dissertation proposed a wireless-based MIDI architecture.

Although several wireless networking solutions were analyzed during this work, *Bluetooth* was the standard chosen to study the implementation of wireless communication between musical instruments.

The proposed architecture benefits from the existing user knowledge on operating *Bluetooth* devices and from the large availability of musical instruments equipped with MIDI interfaces. Besides the improvement on user experience, *Bluetooth* offers several key features when compared to other wireless technologies such as reduced power consumption, high immunity to ISM interference and large commercial availability.

The described architecture allows a direct mapping of MIDI connections on *Bluetooth* links. In this sense, point-to-point wired links are mapped into point-to-point wireless links. In order to transport MIDI command flows using *Bluetooth* links two approaches were implemented and a third proposed. The common feature among all approaches is the use of commercial *Bluetooth* devices, which means that, if any of the approaches is considered for commercial development, only minor changes (or none) will be required.

The first approach used a well-known *Bluetooth* profile, known as Serial Port Profile, to profit from the serial nature of the MIDI protocol. In this sense, a practical test-bed was implemented and test results were obtained. These results showed that MIDI commands experienced high delays that did not cope with the requirements of the application. Notwithstanding, results contributed to the timing qualification of the SPP in noisy environments, which can be used to assess its usability on other industrial applications.

The second approach was developed under the assumption, which became evident later, that the delays experienced by MIDI commands are mostly introduced by the higher-level *Bluetooth* stack. In this sense, the implemented test-bed uses only part of the *Bluetooth* stack, the one included in the *Bluetooth* physical device. The use of a reduced *Bluetooth* stack implies that procedures that were embedded in the stack have to be implemented in a host unit. Therefore, the developed test-bed uses, for each WiMIDI unit, an additional microcontroller that is responsible for establishing and maintaining the *Bluetooth* link according to the requirements of the application.

The developed test-bed has provided results indicating the feasibility of the approach to transport MIDI short commands such as, for example, single notes. However, in a real scenario such as a musical performance, there are chords composed of several single notes that must be played correctly. This approach does not cope with such requirements because it exhibits a high delay for each MIDI command, leading to high variations in the overall delay of a MIDI chord. However, the results may be useful to assess the usage of the approach in noisy scenarios, for example on timing constrained industrial applications.

The approach of using a reduced stack for transmitting MIDI commands exhibited a fair performance for the transmission of short commands. Nevertheless, it does not solve the problem of transmitting chords; reason why a third solution was presented. In this approach, the single note commands, belonging to a MIDI chord, are aggregated into a

single packet and handled as a single note command (as in the second approach). It is expected that the overall delay will be similar to a single note command. Despite the lack of implementation to validate the approach (at the time of this writing) it seems a viable solution because, besides exhibiting a similar delay for single note commands and chords, it does not require any particular synchronization between sending and receiving WiMIDI units.

To summarize, this dissertation presented an architecture for transmitting MIDI using the *Bluetooth* technology. It provided background information on the core technologies involved and described two implementations for achieving the desired results. These results were discussed and, from them, a third approach, capable of supporting MIDI chords, was proposed.

As future work within the MIDI line, it would be important to implement the third proposal in order to verify its validity and, in case of a good performance, to develop a commercial device. Also in this line of work, it would be important to characterize the MIDI data flows in typical musical performances in order to determine the maximum number of supported piconets in a small geographical area such as a stage.

Furthermore, it would also be interesting to evaluate ZigBee as the communication technology since it has recently become commercially available.

Additionally, some outcomes of the work are important for the evaluation of the use of wireless technologies in industrial, time-constrained applications. These also require future work, namely the verification of the impact of different Bluetooth stacks in the timeliness of the solutions.

REFERENCES

[1] Maryanne Cline Horowitz, “New Dictionary of the History of Ideas”, Charles Scribner's Sons, December 2004.

[2] Rodrigues, P., Vairinhos, M, Girão, L., Figueiredo, A., Ferreira, D., Gomes, V., Dias, N., “Integrating Interactive Multimedia in Theatrical Music: the case of Bach2Cage”, ARTECH 2005, 2º Workshop Luso-Galaico de Artes Digitais, August 2005.

[3] Kenton, MIDISTREAM Wireless MIDI System, http://www.kentonuk.com/, April 2005.

[4] Classic Organ Works, MIDIjet and MIDIjet Pro, http://www.organworks.com/, April 2005.

[5] MidiWireless, LIMEX MIDI, http://www.midiwireless.com, September 2005.

[6] Jim Heckroth, “Tutorial on MIDI and Music Synthesis”, MIDI Manufacturers Association, 1995.

[7] MIDI Manufacturers Association, http://www.midi.org, February 2005.

[8] Japanese Association of Musical Electronics Industry, http://www.amei.or.jp, February 2005.

[9] Steven G. Estrella, “Dr. Estrella’s Incredibly Abridged Guide to MIDI”, http://www.stevenestrella.com/midi/, February 2005.

[10] Roland, http://www.roland.com, February 2005.

[11] Pro-Music-News, “Yamaha and Roland agree to cooperate to improve MIDI data compatibility”, http://www.pro-music-news.com/html/01/e10123na.htm, January 2001.

[12] Cakewalk, http://www.cakewalk.com/, June 2005.

[13] Steinberg, http://www.steinberg.de/Steinberg/defaultb0e4.html, June 2005.

[14] RA Penfold, “Advanced MIDI User’s guide”, Second Edition, PC Publishing, United Kingdom, 1996.

[15] Blair School of Music, “Computer Music - MIDI Specification”, Vanderbilt University, http://www.ec.vanderbilt.edu/computermusic/musc216site/MIDI.Specification.html, March 2005.

[16] M-Audio, http://www.midiman.com/, June 2005.

[17] National Association of Music Merchants (NAMM), http://www.namm.com/, June 2005.

[18] Harmony Central Summer NAMM 2001 Coverage, “Bluetooth Wireless MIDI on the Way from Midiman”, http://namm.harmony-central.com/SNAMM01/Content/Midiman/PR/Bluetooth-MIDI.html, Nashville, USA, July 2001.

[19] Lago, N., and F. Kon., “ The quest for low latency”, Proceedings of the International Computer Music Conference, pp. 33-36, Miami, November 2004.

[20] Bluetooth SIG, “Specification of the Bluetooth System 1.1”, Specification Volume 1, February 2001.

[21] Bluetooth Special Interest Group (SIG), “https://www.bluetooth.org/ ¨, May 2005.

[22] IEEE Std 802.15.1- IEEE Standard for Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs), 2002.

[23] Erina Ferro and Francesco Potortì, ¨Bluetooth and WiFi wireless protocols: a survey and comparison¨, IEEE Wireless Communications magazine, pp. 12-26, February 2005.

[24] Ramiro Jordan and Chaouki T. Abdallah, ¨Wireless Communications and Networking: An Overview¨, IEEE Antenna’s and Propagation magazine, Vol. 44, No. 1, February 2002.

[25] Robert Morrow, ¨Bluetooth Operation and Use¨, McGraw-Hill, USA, 2002.

[26] Jaap C. Haartsen, “The Bluetooth Radio System”, IEEE Personal Communications, pp. 28-36, February 2000.

[27] William H. Tranter, Brian D. Woerner, Jeffrey H. Reed, Theodore S. Rappaport and Max Robert, “Wireless Personal Communications – Bluetooth Tutorial and Other Technologies”, Kluwer Academic Publishers, The Kluwer International Series in Engineering and Computer Science, pp. 249-265, 2001.

[28] Chatschik Bisdikian, ¨An Overview of the Bluetooth Wireless Technology¨, IEEE Communications magazine, pp. 86-94, December 2001.

[29] European Telecommunications Standards Institute (ETSI), “3GPP TS 07.10 version 7.2.0 - Digital cellular telecommunications system (Phase 2+) Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol”, 1998.

[30] Bluetooth SIG, “Bluetooth 2.0 Core Specification”, November 2004.

[31] Cambridge Silicon Radio, http://www.csr.com, May 2005.

[32] Broadcom, http://www.broadcom.com, May 2005.

[33] RF Micro Devices, http://www.rfmd.com, May 2005.

[34] T. Jatschka, R. Tschofen, “Bluetooth”, The Industrial Information Technology Handbook, CRC Press, pp.51-1 51-16, 2005.

[35] Bluetooth SIG, “Bluetooth 1.2 Core Specification”, November 2003.

[36] Wi-Fi Alliance, “http://www.wi-fi.com/”, May 2005.

[37] Brent A. Miller and Chatschik Bisdikian, “Bluetooth Revealed”, Prentice Hall, Upper Saddle River, New Jersey, 2001.

[38] Jennifer Bray and Charles F. Sturman, “Bluetooth – Connect Without Cables”, Prentice Hall, Upper Saddle River, New Jersey, 2001.

[39] International 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, IEEE 802.11 WG, ref. no. ISO/IEC 8802-11:1999(E) IEEE Std. 802.11, 1999.

[40] Brian P. Crow, Indra Widjaja, Jeong G. Kim and Prescott T. Sakai, “IEEE 802.11 Wireless Local Area Networks”, IEEE Communications magazine, pp. 116-126, September 1997.

[41] Hua Zhu, Ming Li, Imrich Chlamtac and B. Prabhakaram, “A Survey of Quality of Service in IEEE 802.11 Networks”, IEEE Wireless Communications, pp. 6-14, August 2004.

[42] Matthew S. Gast, “802.11 networks: the definitive guide”, ISBN:0-596-00183-5, O'Reilly, April 2002.

[43] Stefan Mangold, Sunghyung Choi, Guido R. Hiertz, Ole Klein and Bernhard Walke, “Analysis of IEEE 802.11e for QoS Support in Wireless Lans”, IEEE Wireless Communications, pp. 40-50, December 2003.

[44] ZigBee Alliance, http://www.zigbee.org/en/index.asp, December 2004.

[45] IEEE 802.15 WPAN Task Group 4 (TG4), http://www.ieee802.org/15/pub/TG4.html, December 2004.

[46] IEEE Std 802.15.4- IEEE Standard for Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[47] ZigBee Alliance, ZigBee Specification Version 1.0, June 2005.

[48] Ed Callaway, Paul Gorday, Lance Hester, Jose A. Gutierrez, Marco Naeve, Bob Heile and Venkat Bahl “Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks”, IEEE Communications magazine, pp. 70-77, August 2002.

[49] A. S. Tanenbaum, “Computer Networks”, 3rd Edition, Upper Saddle River, Prentice Hall, 1996.

[50] Jianliang Zheng and Myung J. Lee, “Will IEEE 802.15.4 Make Ubiquitous Networking a Reality?: A discussion on a Potential Low Power, Low Bit Rate Standard”, IEEE Communications magazine, pp. 140-146, June 2004.

[51] C. L. Bennett and G. F. Ross, “Time-domain electromagnetics and its applications,” Proc. IEEE, vol. 66, pp. 299–318, March 1978.

[52] Freescale, http://www.frescale.com/, June 2005.

[53] Steve Stroh, “Ultra-Wideband: Multimedia Unplugged”, IEEE Spectrum, pp. 23-27, September 2003.

[54] Domenico Porcino and Walter Hirt, “Ultra-Wideband Radio Technology: Potential and challenges Ahead”, IEEE Communications magazine, pp. 66-74, July 2003.

[55] FCC, “Revision of Part 15 of the Commission s Rules Regarding Ultra-Wideband Transmission Systems”, First Report and Order, ET Docket 98-153, FCC 02-8, February 2002.

[56] ETSI ERM TG 31a, http://portal.etsi.org/erm/ERMtg31A_ToR.asp, December 2004.

[57] Shoie-Chyr Lin, Hsiao-Chiu Chu and Yu-Yen Chung, "A Bandwidth-efficient packet transmission algorithm for Bluetooth scatternets", IEEE ICCS, pp. 824-828, 2002.

[58] 802.15.3, IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs), September 2005.

[59] MBOA-SIG, MultiBand OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a, September 2004.

[60] Wi-Fi Alliance, http://www.wi-fi.org/OpenSection/index.asp?TID=1, December 2004.

[61] WiMedia Alliance, http://www.wimedia.org, December 2004.

[62] Zurbes, S., "Considerations on link and system throughput of Bluetooth networks", Personal, PIMRC 2000, 11th IEEE International Symposium on Indoor and Mobile Radio Communications, London - UK, Volume 2, pp. 1315 – 1319, September 2000.

[63] Duarte P., Fonseca J. A. and Bartolomeu P., "Development and operation of a Bluetooth demonstrator", *Revista do DET*, pp. 519-524, March 2005.

[64] Bartolomeu P., Fonseca J.A., Duarte P., Rodrigues P.M. and Girão L.M., "MIDI over Bluetooth", ETFA 2005 – 10th IEEE International Conference on Emerging Technologies as Factory Automation, Catania - Italy, pp. 95-102, September 2005.

APPENDIX A

LIST OF ACRONYMS

ACK	Acknowledge
ACL	Asynchronous Connectionless Link
AFH	Adaptative Frequency Hopping
AMEI	Japanese Association of Musical Electronics Industry
AP	Access Point
BER	Bit Error Rate
BI	Beacon Interval
BO	Beacon Order
BPSK	Binary Phase-Shift Keying
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CA	Collision Avoidance
CAP	Contention Access Period

CCK	Complementary Code Keying
CD	Collision Detection
CFP	Contention Free Period
COTS	Components of The Shelf
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CTS	Clear to Send
DBPSK	Differential Binary Phase Shift Keying
DCF	Distributed Coordination Function
DIFS	Distributed Inter-Frame Space
DLL	Data Link Layer
DMS	Delay Measurement System
DS	Distribution System
DSSS	Direct Sequence Spread Spectrum
EDR	Enhanced Data Rate
EOX	End of System Exclusive
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FFD	Full-Function Device
FHS	Frequency-Hop Synchronization
GAP	Generic Access Profile
GFSK	Gaussian Frequency Shift Keying
GS	General Standard
GSM	Global System for Mobile communications
GTS	Guaranteed Time Slot
HCI	Host Controller Interface
HR	High Rate
IBSS	Independent Basic Service Set
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
L2CAP	Logical Link Controller and Adaptation Protocol
LAN	Local Area Network
LC	Link Control
LLC	Logic Link Layer
LMP	Link Management Protocol

LR	Low Rate
LSB	Least Significant Bit
MAC	Medium Access Control
MCPS	MAC Common Part Sub-layer
MFR	MAC Footer
MHR	MAC Header
MLME	MAC Layer Management Entity
MMA	MIDI Manufacturers Association
MSB	Most Significant Bit
MSDU	MAC Service Data Unit
MTU	Maximum Transmission Unit
NAV	Network Allocation Vector
NRPC	Non-Registered Parameter Controller
NRPN	Non-Registered Parameter Number
OFDM	Orthogonal Frequency-Division Multiplexing
OSI	Open Systems Interconnect
PAN	Personal Area Network
PBCC	Packet Binary Convolution Coding
PC	Personal Computer
PCF	Point Coordination Function
PDU	Protocol Data Unit
PHY	Physical Layer
PLCP	Physical Layer Convergence Procedure
PMD	Physical Medium Dependent
PN	Pseudo-Random Noise
PPDU	Physical Protocol Data Unit
PSDU	Physical Service Data Unit
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RFD	Reduced-Function Device
RPC	Registered Parameter Controller
RPN	Registered Parameter Number
RSSI	Receiver Signal Strength Indicator
RTS	Request to Send
SAP	Service Access Point
SCO	Synchronous Connection Oriented
SD	Superframe Duration
SDAP	Service Discovery Application Profile

SDP	Service Discovery Protocol
SFD	Start of Frame Delimiter
SHR	Synchronization Header
SMF	Standard MIDI File
SMPTE	Society of Motion Picture and Television Engineers
SO	Superframe Order
SOX	Star of System Exclusive
SPP	Serial Port Profile
SSCS	Service-Specific Convergence Sub-layer
TDD	Time Division Duplex
TDM	Time Division Multiplex
TID	Transaction Identifier
UA	User Asynchronous
UI	User Isochronous
UUID	Universally Unique Identifier
UWB	Ultra-Wide Band
WEP	Wired Equivalent Privacy
WPAN	Wireless Personal Area Network