FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Bluetooth Analysis for Real Time Embedded Sensor Net

João Gomes

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisors: Luís de Almeida, Mario Espinoza

June 25, 2018

Abstract

This document focuses on the technological side of a Bluetooth medical device for prosthetic fitting, herein called the *tInterface* and developed by the Adapttech company, and the need for a technology that can fulfill the device's data transmission requirements while acceptable in the medical device community. A detailed analysis of the product's limitations will be performed, as well as the pursuit of a solution based on commercial off-the-shelf wireless data transmission components. The desired device acquires data about physical parameters, such as pressure and temperature, from the patient-socket interface, sending it via Bluetooth to an Apple iPad, enabling the prosthetist to access detailed information.

At the start of this dissertation's work, throughput and reliability requirements of this product are not successfully met with existing solutions, an analysis into this device and the Bluetooth technology will be performed together with a survey of other possible technological solutions for the same problem. A series of tests and traffic captures allowed us to obtain information about the suitability of technologies to meet the product's requirements, supporting a comparison regarding throughput, power consumption and ease of use between different technologies and specific hardware.

The ultimate goal of this dissertation's work is to find a suitable replacement for the current wireless communication system in Adapttech's product, considering the problem, specific protocols, technologies and solutions involved. The best fitting solution found uses using Bluetooth 4.2 based on a Texas Instruments CC2640R2 System-on-Chip (*SoC*).

Acknowledgements

This work was only possible due to the people and entities involved. I would like to thank Adapttech for giving me the possibility and trust to be involved in their project, more specifically to supervisor Eng. Mario Espinoza, the Hardware/Firmware department leader Eng. André Oliveira and Eng. Frederico Carpinteiro Co-founder of Adapttech. I would also like to thank supervisor Prof. Luís de Almeida, which was very upfront and enthusiastic with the topic of this dissertation from the beginning, for the guidance throughout this last months. A special acknowledgment to my girlfriend, to my parents, brother and family who have been the central pillar of support throughout my academic path.

João Gomes

"With great power, comes great responsibility."

Voltaire

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ACL	Asynchronous Conection-less
ADC	Analog-Digital Converter
ADVB	Advertisement Broadcast
AES	Advanced Encryption Standard
AFH	Adaptative Frequency Hopping
AMP	Alternate MAC/PHY
API	Application Programming Interface
ARQ	Acknowledge Repeat-Request
ATT	Attribute Protocol
BAN	Body Area Network
BLE	Bluetooth Low Energy
BR/EDR	Basic Rate / Enhanced Data Rate
BT	Bluetooth
CSRK	Connection Signature Resolving Key
DLE	Data Length Extension
GAP	Generic Access Profile
GATT	Generic Attribute Protocol
HCI	Host Controller Interface
HDP	Health Device Profile
HS	High Speed
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Motion Unit
IP	Internet Protocol
IRK	Identity Resolving Key
ISM	Industrial, Scientific and Medical
LL	Link Layer
LMP	Link Manager Protocol
LTK	Long Term Key
L2CAP	Logic Link Channel Adaptation Protocol
MAC	Medium Access
MFi	Made For iPod
MICS	Medical Implant Communication Service
MCAP	Multi Channel Adaptation Protocol
OOB	Out of Band
OSI	Open System Interconnection
PADVB	Periodic Advertisement Broadcast
PHY	Physical Interface/Hardware
PDU	Protocol Data Unit

RFCOMM	Radio Frequency Communication
SCO	Synchronous Connection Oriented
SD	SecureDigital
SDP	Senvice Discovery Procotol
SDU	Service Data Unit
SIG	Special Interest Group
SM	Security Manager
SPP	Serial Port Profile
SYM	Symbol
TCP	Transport Control Protocol
TDD	Time Division Duplex
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
UWB	Ultra Wideband
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WRAN	Wireless Regional Area Network
WTMS	Wireless Medical Telemery System

Chapter 1

Introduction

Adapttech is a startup company that focuses on the design of adaptation technologies —any type of technology to help people with physical limitation to improve their quality of life. One of their solutions helps patients go through a faster and easier prosthetic fitting process. The motivation for this dissertation arises from a specific need that Adapttech has to improve: wireless medical device communications.

In the last decades, the world has witnessed an increase in percentage of amputee population. In the 1950s causes for lower-limb amputations were due to work accidents, but the recent high-growth trend is actually explained by vascular diseases caused mainly by diabetes. The proliferation of such diseases is directly connected to the change in living habits that occurred specially in the Western Hemisphere.

Prosthesis have evolved using now advanced and smart materials, and much research has been done in that specific field. Despite this evolution, the process of handcrafting a customized socket adequate to each patient's amputation still requires improvements. Most patients need several appointments and prosthesis adjustments before it becomes comfortable enough to be used every day. Adapttech aims to solve this problem by creating a prosthetic-fitting tool that helps the technician to determine how the pressure is distributed between the stump-socket interface.

Therefore, this aim is to contribute to the improvement of this process and facilitate the adaptation of prosthesis to those in need. Particularly, this can be done by assessing the prosthetic fitting, which is the scope of the present work.

This chapter introduces Adapttech and its products, as well as the problems that we will address in this work and the proposed solution we developed.

1.1 Context

Adapttech's main mission is "To develop biomedical technologies to help people with physical limitations improve their quality of life".

Within that mission, Adapttech is currently developing three products, namelyt, the *tLaser*, *tInterface* and *tAnalyzer*. Adapttech has formulated a solution to make the prosthetic fitting process

faster and more reliable for amputees. As the patients themselves have trouble communicating exactly where the problem with their prosthetic is, the solution found by Adapttech is to create a 3D model and then, embedding a network of pressure sensors, enabling the prosthetist to have a more accurate feedback of what area in the socket is causing discomfort to the patient. The sensor data is displayed to the technician in an Apple iPad application, overlaying the 3D model. The *tInterface* is, as of the date of writing this document, a wearable device that is connected to the sensor network, collects its data and communicates with an application via a Bluetooth 3.0 module . The system was designed to host a network of up to 128 sensors, and each sensor is sampled with an 8-bit Analog-Digital Converter (ADC). As the sample rate for the network is 50 Hz and having a byte of data per sensor, the throughput required to display data in real time is 71,8 kb/s. Sensors are grouped in strips containing 8 sensors each. This wearable is also provided with 2 nine axis inertial motion units (IMU), featuring an accelerometer, gyroscope and magnetometer, being also able to perform a gait analysis of the patient, and thus map the pressure data in a given point in time, to the position of the leg. Two uses cases were considered for this device, for clinic appointments between patient and prothetist, and to collect and store data offline for post analysis. This process is illustrated in Fig. 1.1.

Among Adapttech's products *tLaser* is the device which performs 3d scans prosthetic sockets(Fig. 1.2), creating the respective 3d mesh for each prosthetic. This device is composed of a laser, 2 cameras, and the socket support. A full scan starts with the laser being projected inside of the prosthetic socket, while the cameras record the laser's position. By taking several photographs at different heights, a master computer is able to generate a 3D mesh. A second scan is then done, without the laser, with the purpose of identifying sensor strips that were previously inserted into the socket. This device plays a fundamental role in the smart prosthetic fitting solution, as the 3d mesh will serve for a reference to identify the problems derived from the socket/limb interface.

The *tInterface* is a wearable embedded system, built in a rigid-flex PCB, responsible for sampling the pressure sensor strips (Fig. 1.3) applied into the patient's socket. This system is currently composed of a microcontroller, an Atmel SAML21J18B to communicate with the ADCs and generate the acquisition data application message, a Microchip BM78 Bluetooth module for the communication with the mobile application. Each pressure sensor strip, referred as *tStrip*, has its own PCB with a circuit to linearize to the sensor output, and an ADC to collect data. Bluetooth communication is done using an RFCOMM channel between an iPad and the tInterface, creating a virtual serial port between both devices, enabling an abstraction from all of Bluetooth's protocols, as the communication system is seen as a *black box*, implementing only an UART tunnel, from a developers perspective.

After collecting data from the sensors, data is displayed to the user in the *tAnalyzer*, an iOS application(Fig. 1.4). The application uses the patient data to show the 3D mesh with the overlapping sensor locations. Different pressure values result in differences in the colour of the 3d mesh, in that sensor's location, providing to the technician reliable data to be able to evaluate and adapt the prosthesis accordingly.

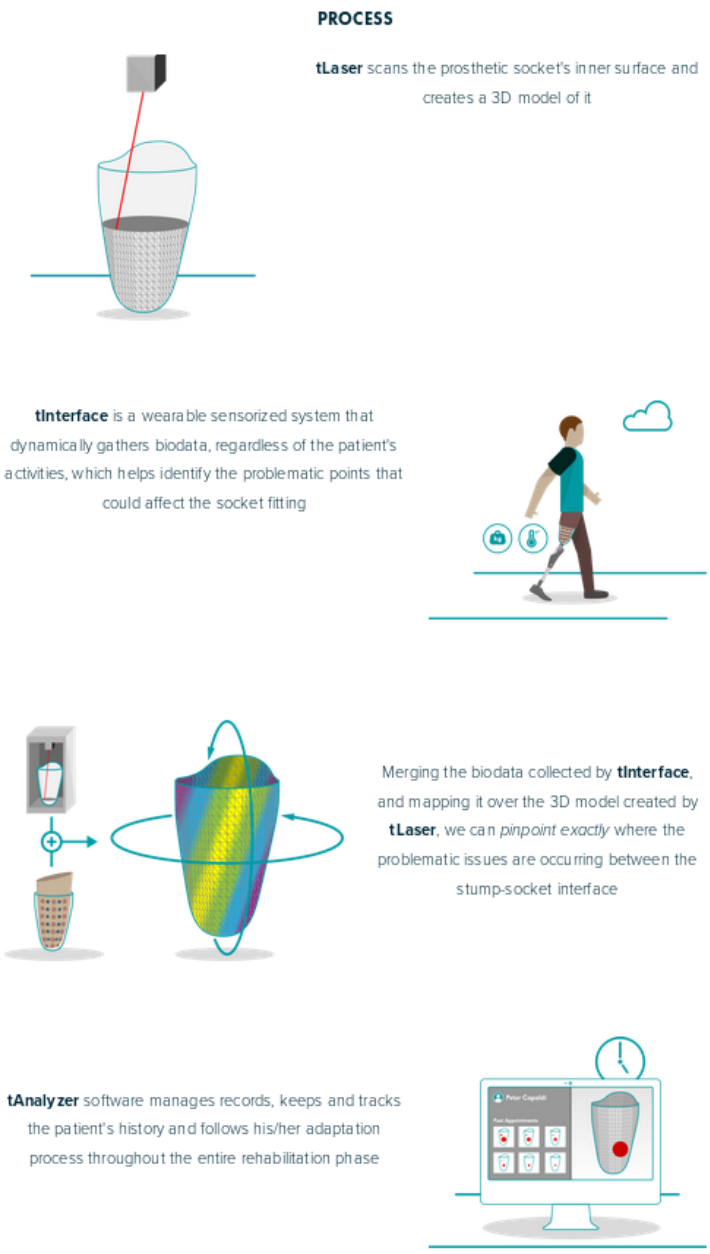

Figure 1.1: Adapttech's process illustration

1.2 Objectives for this work

The objectives for the work developed in the scope of this dissertation are to contribute to the instrumentation of prosthetics within the scope of tInterface as described in the previous section. The minimal technical requirements defined for the tInterface system are:

- A minimum sampling frequency of 50Hz.

Figure 1.2: Simple tLaser illustration

Figure 1.3: Simple tInterface illustration

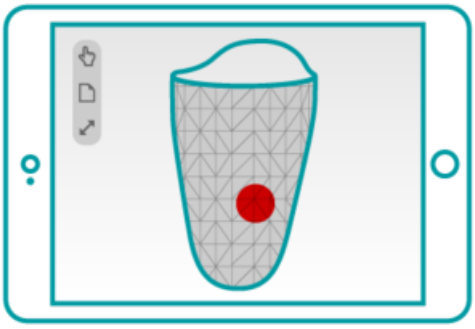

Figure 1.4: Simple *tAnalyzer* illustration

- Support 128 pressure sensors and 2 inertial motion units, totalling 164 bytes of information per sampling cycle.

- Be able to communicate with iPad device, which is approved by the FDA (U.S. Food and Drug Administration) as a medical device.
- To be as low power as possible
- Maintain user interaction and configuration as simple as possible.

The primary objective is to be carried out is to achieve sufficient throughput in a reliable way between the embedded system and the iPad, effectively enabling the product to transmit more data. More specifically, the aim is to study and understand the protocols and specifications used in this system to communicate, supporting the development and testing of a prototype of modules to, that is capable to achieve a reliable sensors sampling at 50 Hz. Implementation should be done with minimal changes to the product characteristics while consuming the least energy possible. Although it is not an engineering requirement, using an iPad tablet is imperative to Adapttech as a business requirement which raises some issues that will be explored.

1.3 *tInterface* Description

This dissertation is focused on this device and the iPad it communicates with via Bluetooth, as such, we present next a more detailed explanation of how this device works and how it is composed.

The *tInterface* device is composed of two separate entities. The rigid-flex part, composed of USB-C female connectors, where the sensor strips are connected by the user, and the main board, where the Bluetooth module and SAM microcontroller, battery charging ICs and one MPU 9250 Inertial Motion Unit are present. Besides reporting pressure data, this device also provides data from 2 IMUs in order to support a gait analysis algorithm, based of machine learning, that runs in a remote backend. With this in mind, the recommended sampling periods, for the sake of compatibility with previously acquired data, are constrained to specific values. A sampling period of either 20, 15, 10 or 5 ms are the values the recommended by Adapttech's Bio-Engineering Department. The information is collected by the microcontroller in the *tInterface* wearable system from the pressure sensor network resorting to Analog-Digital Converters, embedded in each sensor strip . The currently employed solution features a Texas Instruments AD088S052, capable of a sampling rate up to 500k samples per second. An abstract system diagram can be seen in Fig.1.5.

In the initial connection setup between the *tInterface* and the iPad, the iPad sends a message requesting information about the sensors present in the system. After this information is correctly exchanged, a message is sent to the *tInterface* for it to start sampling the sensors and sending information in real-time. A simple protocol is used to support this communication, employing a message header and tail, in order to support different message types and lengths, as well as verifying message integrity. As data is effectively sampled at a minimum rate of 50Hz, and considering that up to 128 sensors may be in the sampling loop, a quick calculation is enough to quantify the required throughput for application data, is 71200 bits per second, considering a maximum application message length of 177 bytes.

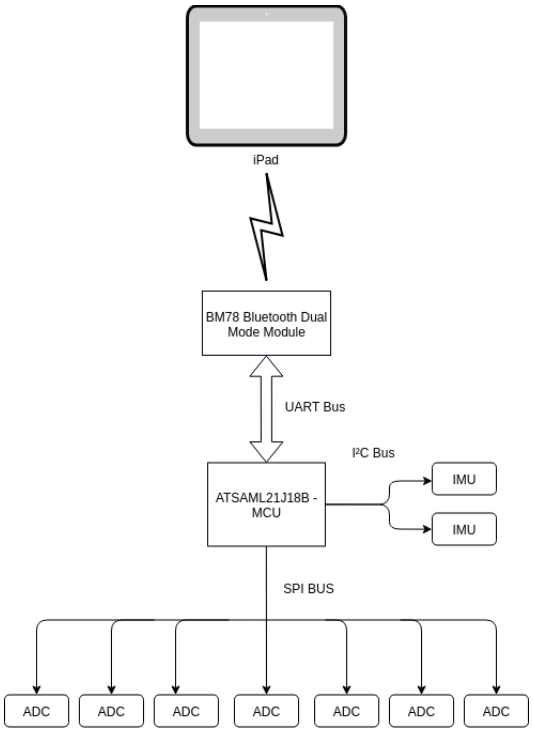

Figure 1.5: System Diagram

1.4 Dissertation Structure

This chapter introduced the context of this dissertation and laid down the objectives for this work. Chapter 2 presents an overview of some popular wireless communication technologies and standards will be given, followed detailed view of the problems regarding the *tInterface*. Chapter 4 gives a description of Bluetooth technology, as it is the current solution employed in the product to communicate wirelessly with the iPad. Chapter 5 presents the series of tests performed with different Bluetooth modules and *Socs* and the respective results. Chapter 6 presents a study regarding the possibility of a second external IMU system. Finally, Chapter 7 summarizes the problem, solution and the results that were achieved, considering the main the theoretical and practical aspects discussed throughout the document. This chapter also presents a recommendation regarding the wireless technology that is able to fulfill requirements - Bluetooth Low Energy - and and discusses which would be the best solution for Adapttech's *tInterface* product.

Chapter 2

Literature review

2.1 Medical Device Communications

As technologies evolve, so do medical devices and, in the current days, possibilities for smarter technologies for monitoring and analyzing vital signs and biological information have emerged, providing ever improving quality of life and establishing new standards. For a long time technology has been involved in a day to day basis with human life. As cheaper and more precise devices appear, usability becomes of significant importance to the proliferation of a certain technology. Many of the medical devices available today feature wireless communication standards, allowing for patients and medical personnel to have more practical approaches to solving and determining problems. [7]

Telemedicine is one of those examples, providing a method for supervisory control between both parties, without requiring physical proximity. Body Area Networks (BANs) are also one of the concepts that medical professionals today, as well as technology companies, invest in to give them the ability to monitor different bio-signals resorting to different sorts of sensors.[8]

Body sensor networks have emerged, causing new challenges in the industry and introducing new and ever more specific features and requirements for technologies.[9]

Wireless communication in medical devices predominantly use the Industrial Scientific and Medical (ISM) frequency band, which is unlicensed, although various exceptions exist. There is no standard specifically defined for medical devices BANs, as a result several technologies such as GSM, Wifi, ZigBee, Bluetooth and Ultra Wide Band (UWB) are used. In addition to the ISM unlicensed band, there are bands defined for medical devices in the spectrum such as Medical Implant Communication Service (MICS) and Wireless Medical Telemetry Service (WTMS), bands that support short and long range communications. One of the advantages of these licensed bands is that there are less interference possibilities, given that only licensed equipment may operate on those frequency bands.

The most important characteristics for medical wireless devices are, among other, power consumption, electromagnetic compatibility, interoperability with other devices and physical characteristics such as size. Table 2.1 presents an overview over some of the more widespread wireless

communication technologies and bands used in medical devices.

	MICS [6]	WMTS [6]	UWB IEEE (802.15.6) [28]	IEEE 802.15.4 (ZigBee)	IEEE 802. 15.1 (Bluetooth)	WLANs (802.11b/g)
Frequency band	402–405 MHz	608–614, 1395–1400, 1429–1432 MHz	3–10 GHz	2.4 GHz (868/915 MHz Eur./US)	2.4 GHz	2.4 GHz
Bandwidth	3 MHz	6 MHz	>500 MHz	5 MHz	1 MHz	20 MHz
Data rate	19 or 76 kbps	76 kbps	850 kbps to 20 Mbps	250 kbps (2.4 GHz)	721 kbps	>11 Mbps
Multiple access	CSMA/CA, polling	CSMA/CA, polling	Not defined.	CSMA/CA	FHSS/GFSK	OFDMA, CSMA/CA
Trans. power	−16 dBm (25 μW)	≥10 dBm and <1.8 dB	−41 dBm	0 dBm	4, 20 dBm	250 mW
Range	0–10 m	>100 m	1, 2 m	0–10 m	10, 100 m	0–100 m

Table 2.1: Bands used in wireless medical devices

2.2 IEEE Network Standards

The Institute of Electrical and Electronics Engineers Standards Association has published and is actively engaged in developing, standards for networking to assure interoperability, organization and coexistence between communication technologies.

Several wireless standards are defined by IEEE, providing an organized and systematic approach to generic types of communication. This methodology is necessary in order to guarantee reliability, environmental compatibility, and structured standards for various types of applications.

The OSI model is an important reference, enabling technology to be developed in a decentralized layered approach. This way it becomes easier for protocols from different layers to interact with one another. An abstract protocol stack is defined with the OSI model, defining how interaction between different layers is able to deliver data from applications up to the physical media.

Various types of wireless networks can be found:

- IEEE 802.22 (WRAN)- Wireless Regional Area Network is intended for standards regarding long range communication, for devices that have operating needs to communicate further than regular urban distances.

- IEEE 802.16 (WMAN) - Wireless Metropolitan Area Network standards define operation for inner city networks.[10]

- IEEE 802.11 (WLAN) - Wireless Local area networks are very widespread, normally used for communication within buildings. They have medium range, and are supported by most mobile devices succh as laptops and cellphones.

- IEEE 802.15.1 (WPAN) - Wireless Personal Area Networks are intended to support operation between devices on a very short range (<10m).

- IEEE 802.15.6 (WBAN) - Wireless Body Area Networks mostly used for medical/biomedical applications that retrieve information about vital signs from the human body.[11]

2.3 OSI Model

This model is a standard reference in the communications world, as it provides guidelines for development of network protocols and applications. The model is composed of a layered structure, in order to improve reliability and interoperability, and facilitate implementation. Its ratification happened in 1984, and although the protocol does not indicate how physical parts should be implemented, instead provides an overview on the techniques to build a structured network, assigning different tasks and responsibilities for each layer. [12]

Although this model became a standard, it only indicates good practices, and implementations can vary, e.g., the TCP/IP protocol follows a different organization of layers. Different technologies may or may not apply this model, but in a general way this model is embedded into the mindset of the electrical and computer engineering world and most protocols are designed with this model in reference.

The model defines a seven layers structure with an associated terminology represented in Fig. 2.1.

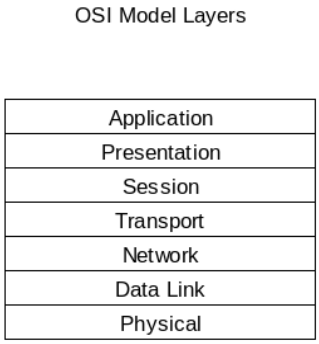

Figure 2.1: OSI Model Definition

2.4 Bluetooth Technology

> "Bluetooth wireless technology is a short-range communications system intended to replace the cable(s) connecting portable and/or fixed electronic devices. The key features of Bluetooth wireless technology are robustness, low power, and low cost." [2]

Bluetooth technology was created in the late 90's, by Ericsson, Intel and Nokia. Its main goal was to provide simple interoperability between devices in a PAN (Personal-Area-Network) without the need for cables. It was intended to be used with printers, mobile devices, computers, etc. Since its early days the technology has evolved and expanded into a globally accepted standard available in almost every mobile device today. Bluetooth has classically been used to replace serial ports, creating a virtual serial port in both of the devices, allowing for backward compatibility with older devices.[13]

Bluetooth's simple user interaction and seamless interoperability between devices, supporting multiple and diverse functions, is only possible due to the numerous profiles, and protocols involved. Below the simple user interaction lies a complex specification, developed with a generic approach to applications, in order for it to be able to support ever appearing new features and applications.

In 1998 the Bluetooth Special Interest Group (SIG) was created with 5 founder companies - Intel, Ericsson, Nokia, Toshiba, IBM - and one year later the Bluetooth 1.0 Specification was released. From then on, the number of supported devices only grew. On the following years Bluetooth continued to be a supported technology, with continuous improvements, as new specifications were released.

As it became a widespread standard, the initial purpose of replacing cables for connecting devices hasn't changed, but requirements for throughput, distance, pairing time and power consumption have all become stricter, and Bluetooth's applications (use cases) ever more demanding. Today, the several specifications enable the possibility for many different types of devices, from the really low power devices that can run for a year on a coin cell battery, to mesh networking devices, audio devices such as headsets and speakers. Nowadays Bluetooth is a mature technology with 3 different physical controllers, Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR), Bluetooth AMP (AMP) Bluetooth Low Energy (LE), in order to cater for specific requirements. Both systems, BR/EDR and BLE, feature services for device discovery, pairing and connecting. Whilst Basic Rate/Enhanced Data Rate is intended to be used on data intensive applications, the other, Bluetooth Low Energy, is generally used in cases of low frequency data sensors with reduced energy and production costs. Shipped Bluetooth devices are over 3.4 billion per year, as the technology is a proved solution for the problems that it was intended to solve.[14] The availability and flexibility of this technology makes it very appealing for developers and users. Bluetooth devices are composed a host and a controller, using a series of links and channels, to effectively transmit data.

With each specification new features are introduced, an overview of the most relevant features will be highlighted in this section.[2]

- v1.2
  - Adaptative frequency hopping
  - Extended SCO links
- v2.0 + EDR
  - 2 Mb/s and 3 Mb/s PHY
- v2-1 + EDR
  - Erroneous Data Reporting
  - Extended inquiry response

  - Secure Simple Pairing

- v3.0 + HS
  - L2CAP Enhanced Retransmission and Streaming Modes
  - AMP
  - 802.11 PAL
  - USB and SDIO HCI transports

- v4.0
  - LE PHY, Link Layer
  - AES encryption
  - Attibute protocol
  - Generic Attribute Profile

- v4.1
  - BR/EDR secure connections
  - LE Dual Mode Topology
  - LE L2CAP connection oriented channel
  - LE Ping

- v4.2
  - LE data packet length extension

- v5.0
  - LE 2 Msym/s PHY
  - LE Long Range

2.5 WiFi

WiFi technology arises in a general way in the early 2000's, quickly becoming a very promising technology. Support, evolution and new specifications have made possible communication for devices within a household, bar, hospital, to work in a cordless way. This has impacted profoundly our societies behaviour patterns, changing them significantly and introducing new habits, technologies and technological expectations.

WiFi is a commercial brand for the Wireless Ethernet Compatibility Alliance, and its certified products. This organization was created with the purpose of assuring interoperability and integration of 802.11 based norms, also providing a specific trademark making the technology's marketability more appealing.

Wifi can be abstractly placed in the WLAN standard, being a low-medium range wireless communication technology. The technology itself is composed of several norms and protocols, defining interaction between different layers that compose it. The two layers that are defined are the physical PHY and MAC layers, responsible for physical data transportation and defining how different nodes have access to the medium.

Several PHY norms have been designed and implemented over the years, improving connection bandwidth and radio related problems. Today, the 802.11n norm is the most used. Also recently 802.11ac has been entering the market, being incorporated by manufacturers in their designs.

The numerous norms differ primarily in the modulation scheme used to transmit data, being a direct factor in the frequency of operation, range, interference and obviously, throughput. A comparative chart regarding such aspects is presented in Fig. 2.2.

Protocol	Frequency	Channel Width	MIMO	Maximum data rate (theoretical)
802.11ac wave2	5 GHz	80, 80+80, 160 MHz	Multi User (MU-MIMO)	1.73 Gbps[1]
802.11ac wave1	5 GHz	80 MHz	Single User (SU-MIMO)	866.7 Mbps[1]
802.11n	2.4 or 5 GHz	20, 40MHz	Single User (SU-MIMO)	450 Mbps[2]
802.11g	2.4 GHz	20 MHz	N/A	54 Mbps
802.11a	5 GHz	20 MHz	N/A	54 Mbps
802.11b	2.4 GHz	20 MHz	N/A	11 Mbps
Legacy 802.11	2.4 GHz	20 MHz	N/A	2 Mbps

Figure 2.2: Comparrison between several 802.11 PHY [1]

2.6 Ant

> "ANT is a 2.4GHz bidirectional wireless Personal Area Network (PAN) communications technology optimized for transferring low-data rate, low-latency data between multiple ANT-enabled devices. " [15]

Ant is a wireless technology enabling the use of low power sensors. It defines protocols for the 3 lower layer of the OSI model. Higher layer is to be defined and implemented by the user. This technology is intended to provide simple data transmission for low cost microcontroller units, providing for cheap and low power devices. An example of the stack layering from the ANT specification is presented in the next figure. Supported hardware is manufactured by Nordic Semiconductors, such as modules, USB sticks and development kits.

Ant communications are based in the ISM spectrum, being unlicensed.

One of ANT's relevant characteristics is the low data rate, making it unfit for applications with higher performance requirements. Its maximum throughput is 60 Kb/s.[15]

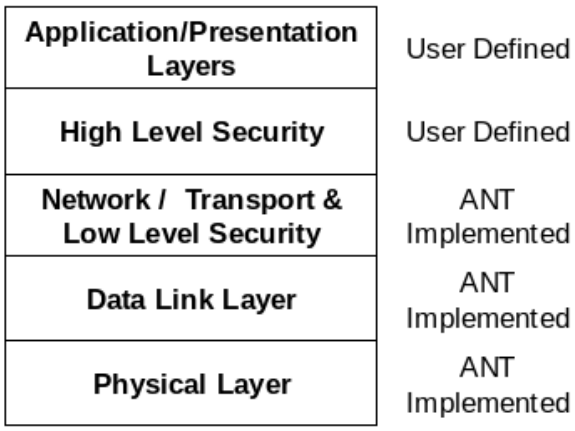

Figure 2.3: ANT stack

2.6.1 Physical Layer

ANT's physical layer operates on the ISM 2.4GHz band,, dividing a 124MHz frequency range into 125 physical channels. Medium access is governed by a TDMA scheme, allowing for multiple channels to coexist.[16]

2.6.2 Data Link Layer

Data is transported in channels, that need to be configured and equal in both endpoints, in order for information to be transported. The protocol allows sending bursts of data, improving low power capabilities.[17]

2.6.3 Transport Layer

Data between the ANT hardware and its host microcontroller is exchanged via serial communications, obeying a specific message structure, illustrated in the following image.

Figure 2.4: ANT message

The hardware bus itself is not specified, so implementations may vary for different applications/manufacturers.

2.6.4 Network Layer

ANT networks support diverse configurations allowing for easy interaction between nodes and information dissemination. Networks can be decentralized and have complex topologies. Connections are bi-directional and basic networking is enabled by using different channels to connect different nodes.

2.7 Zigbee

> "Wireless sensor networks are an emerging technology for low-cost, unattended monitoring of a wide range of environments. Their importance has been enforced by the recent delivery of the IEEE 802.15.4 standard for the physical and MAC layers and the forthcoming ZigBee standard for the network and application layers." [18]

The Zigbee protocol, proposed by the Zigbee Alliance, has influenced heavily sensor networks. It defines a network layer that specifies several network topologies. Zigbee allows for large quantities low power devices, sampling and transmitting data, to be applied in diversified fields such as agriculture, inventory monitoring, motion tracking and many other. Common Zigbee devices are composed of a ordinary 8 bit microcontroller, sensors and a 802.15.4 transceiver. The availability and cheap cost make this technology an attractive solution for many use cases. The radio operation is performed in the ISM band, support data rates up to 250 kb/s and a maximum range of 300 meters.[18] The IEEE 802.15.4 defines a physical layer and a MAC layer for Low Power Wireless Personal Area Networks (LR-WPAN).

2.8 UltraWideband

> "Ultra-wideband (UWB) technology offers a solution for the bandwidth, cost, power consumption, and physical size requirements of next-generation consumer electronic devices"[19]

Ultra wideband, formerly known as pulse radio, is a wireless technology, offering a diverse range of solutions such as radars that can identify objects through walls or floor, distance measurement and communication. The technology can theoretically be used for WPANs with high bandwidth, supporting data rates greater than 100Mbit/s. UWB emits radiation over a very wide frequency range, possibly overlapping with existing services. Frequency is controlled through pulse shaping and several modulation techniques can be implemented, having impact over power considerations and in direct relation with possible interference caused to telecomunication systems. UltraWideband communications use a recently approved frequency, from 3.1GHz to 10.6Ghz. This allows for a very big range of frequencies, providing a new approach to wireless communication technologies.[19]

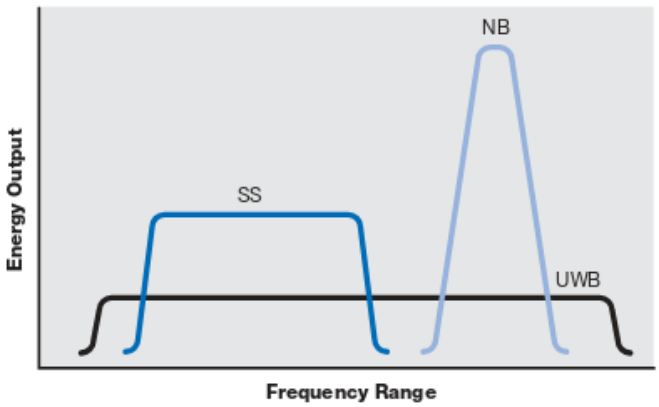

Figure 2.5: UWB frequency range comparrison

A startup company, Decawave has already developed cost effective modules featuring Bluetooth Low Energy and UltraWideband technology, in order to have Real Time Location Services (RTLS).

> "DecaWave's DW1000 chip, is a complete single-chip CMOS Ultra-Wideband IC based on the IEEE802.15.4-2011 standard. DW1000 is the first in the DecaWave ScenSor family of parts, operates at data rates of 110 kb/s, 850 kb/s and 6.8 Mbps, and can locate tagged objects both indoors & outdoors to within 10 cm.[20]"

However, UWB is still just a promising technique, where its capabilities have been explored very little, as it is technology in its early stages. Protocol stacks are already envisioned, as can be observed in Fig 2.6.

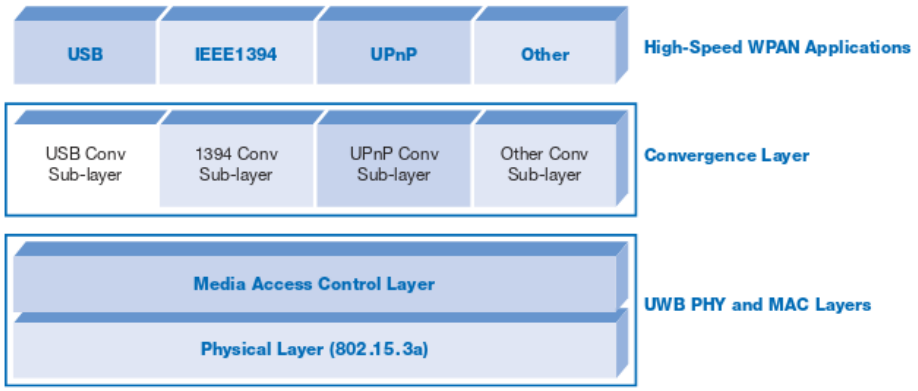

Figure 2.6: UWB Protcol Stack

2.9 Summary

After a brief exploration the previous technologies, a qualitative table regarding the most relevant aspects in the reviewed technologies is presented.

	Power Consumption	Maximum Throughput
Bluetooth	Low/Medium	Medium
ANT	Low	Low
UWB	Low/Medium	High
WiFi	High	High
ZigBee	Low	Low

Table 2.2: Wireless technology power and throughput comparison

Several factors will need to be taken into account when choosing a wireless technology for the *tInterface*. Although throughput and power consumption are important factors when choosing a technological solution for this problem, the user experience also has to be considered. As usability is an important feature on a wearable product such as the *tInterface*, wireless technologies that imply more complex configuration procedures for the user are less desirable. The simple interaction provided by Bluetooth was an important factor when considering the technologies that will be subjects of interest for further work. A network with a fixed access point is also not appealing when considering the application in question, as the devices does not have any keyboard or inputs, configuration of an WiFi AP would be an issue and also the fact that the device would only work in the range of this AP. Also, as Bluetooth Low Energy permits the creation of lower power devices, it seems a viable technological solution for this problem, as it is currently used in many other wearable devices. Furthermore, Bluetooth Low Energy is a technology that has seen good evolution in the last years and is a technology with an actual and competitive market. The specific need of communication with an iPad was also a defining requirement when choosing an appropriate wireless technology, as Apple's environment is usually restrictive and not all technologies considered can interact with its hardware, such as ANT, ZigBee and UWB. The Bluetooth specification supports the data rates required by the application and is present in every Apple iPad tablet, thus being a technology with characteristics that can fulfill all of the `tInterface's` communication and usability needs. WiFi also supports the required throughput capacity, exceeding it by far. However, in terms of usability and configuration does not appear to be fully adapted to the application in question.

Chapter 3

Problems and Solutions

3.1 Problem Definition

The problem to be addressed in the dissertation work is the study of low power high throughput wireless technologies to find a suitable solution for the *tInterface* device of Adapttech. As configured and used prior to this work, the throughput provided did not allow for a satisfactory achievement of the defined requirements.

When in operation, the *tInterface* sends messages from the Bluetooth module to the iPad, those messages include, among other fields, a 4 byte timestamp provided by the microkernel running in the *tInterface's* microcontroller. When the application is running, pressure data received can be observed represented by a colored mesh overalapped over the 3D scan of the prosthesis with a color range with 256 values. When inspecting in detail the application's debug output and, although visually data is apparently flowing correctly, the message timestamps have frequent gaps of about 280 milliseconds to 1 second. This problem, reported by Adapttech, was first discovered by the Software Department when testing of a program for internal use, which revealed the lost messages and the unreliability of the connection. These gaps are, in fact, a crippling problem to Adapttech's application, as it makes it impossible for any machine learning algorithm to be accurate with the amount of data that is missing.

The messaging protocol used to transfer information between both endpoints, as shown in Fig. 3.1, is composed of a message header with 3 octets, 2 length octets, a variable size payload and 3 other octets in the end closing the message.

Figure 3.1: Application Packet Structure

The algorithm for receiving messages checks for the header, receiving a variable number of bytes determined by the length field, and after checks to see if the last 3 bytes match a predetermined sequence. Although this technique was implemented to check consistency in messages,

some unnecessary overhead is created with the usage of that protocol, as the last 3 bytes are redundant and do not enable any forward error correction or cyclic redundancy check. The algorithm is designed to determine if data in a specific use case is correct: A packet header is received containing length information, then the rest of the message is received and the tail sequence confirmed. If the tail sequence is not correct, the message is probably missing some bytes in the middle and therefore is discarded as intended.

Microchip's BM78 Bluetooth module embedded in the system is the entity responsible to receive data from the microcontroller and transmitting it wirelessly to the iPad tablet. The module is configured to use the Serial Port Profile in order to be accessed as a virtual serial port. This profile enables for simple interaction and minimal programming on the application side, where the program only needs to access an OS virtual serial port, and as it is an old and well established concept it is relatively cost and time efficient to implement. The BM78 provides a simple interface with the microcontroller, using a transparent UART tunnel from the system to the iPad. Microchip only provides a computer program to change the Bluetooth parameter options, but that can never be done "on-the-fly", and only SPP and iAP profiles are supported.

One critical problem is evidenced in the module's datasheet, as it does not offer support for the required data rates, as seen in Fig.3.2. The required application throughput is 71kbit/s, while this module provides 32kbit/s at most. The Bluetooth specification, although defining every protocol exhaustively, does not describe how the stack should be specifically implemented, just how it should behave. Although compliance testing is required, many of the features specified do not have to be implemented, being optional to the manufacturer. As one cannot interact with the stack running in the module or have knowledge on how it is implemented, no conclusions can be made leaving many questions and uncertainty regarding the possible consequences related to implementation details.

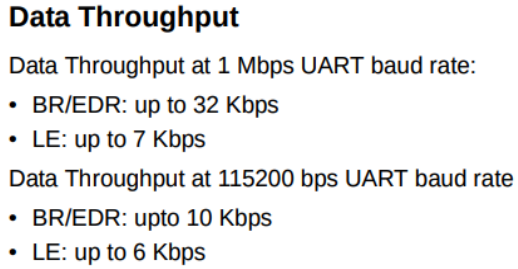

Figure 3.2: BM78 maximum supported data rates

Another problem is that, in order to communicate with an iPad using BR/EDR, it is required using iAP, Apple's Accessory Protocol. In a forum post for a Bluetooth module manufacturer, Bluegiga, its support briefly discusses this subject.

> "The iAP protocol has a lower maximum throughput than a pure RFCOMM/SPP connection due to the way it is designed." [21]

As MFi (Made For iPod) Program and associated documentation can not be discussed or disclosed and no successful BR/EDR capture was achieved, there is no straightforward explanation as to why that occurs.

Part of the problem is derived from the tablet side since the tablet initiates the connection. Thus, it inherently assumes master role in the connection, controlling the connection parameters. The API provided by Apple for Bluetooth related programming does not support changing connection parameters as the OS manages them internally. An aggravating factor is the lack of information provided by Apple regarding iAP(Acessory Protocol) protocol and its kernel management of Bluetooth connections.

Another factor restraining communication is that Bluetooth 2.1/3.0 devices require specialized hardware/firmware in order to be able to connect with Apple iPod, iPhone and iPad. This external hardware is responsible for authenticating the accessory as an authorized MFi product, introducing overhead to each communication packet, and limiting throughput.

> "To incorporate iAP into an accessory design, the accessory developer must be a member of the Apple MFi licensing program and integrate specific MFi hardware into the accessory." [22]

As either of the endpoints is incapable of providing an interface for using different profiles or changing meaningful connection aspects, one is not able to change in any advantageous way this Bluetooth connection.

3.2 Proposed Solutions and Improvements

The product developed before this work does not does not meet the full requirements, providing communication with a certain degree of service quality, only. However, we believe that the Bluetooth technology is able to support and meet all the requirements, if different connection parameters and configurations are used. In the last years, a wide popularity of iPads for medical applications has emerged, making the device a very marketable approach for companies and having advantages regarding development costs and know-how, when compared to developing a fully customized system. Therefore, a solution maintaining the Bluetooth technology and the iPad tablet device is in the best interest of the company.

Since the communication cannot be monitored at either endpoint, we decided to use a traffic sniffer to provide an analysis of the packets in the air. Traffic analysis is an invaluable tool in offering detailed data about how the two endpoints are specifically interacting, and being able to understand what and how stack layers are being used as well as roles, maximum transmission units for transport protocols and providing accurate throughput information.

3.2.1 Traffic Analysis Tools

As most Bluetooth modules do not allow promiscuous mode captures, capturing Bluetooth traffic is not as easy as with other technologies such as WiFi, in which a good percentage of network cards

allow a promiscuous mode, in which air packets can be captured. As bluetooth uses frequency hopping, a technique that requires consecutive packets to be transmitted in different frequencies, capturing packets when not synchronized with the hop pattern is not possible, so software based approaches for analyzing Bluetooth traffic like Wireshark are only relevant if the connection being analyzed is with the computer running the packet analyzer, otherwise monitoring is almost impossible. Hardware solutions generally have a high price, but there are hardware-based traffic-analysis tools that are affordable such as is the open-source Ubertooth project.

> "Project Ubertooth is an open source wireless development platform suitable for Bluetooth experimentation. Ubertooth ships with a capable BLE (Bluetooth Smart) sniffer and can sniff some data from Basic Rate (BR) Bluetooth Classic connections."[23]

Although not all data traffic from Classic Bluetooth connections can be sniffed, the Ubertooth is already a good help. However, to capture and analyze traffic to the smallest detail, professional tools are required.

3.2.2 BR/EDR L2CAP direct transport

Using an L2CAP connection for data transport, control over the transmission features such as retransmission and flow control can be obtained, also possibly containing less overhead than if using RFCOMM, and therefore enhancing efficiency. To reduce the number of layers involved, there is the possibility to use an L2CAP connection oriented-channel, without having unnecessary RFCOMM encapsulation and signaling and control messages creating unnecessary overhead.

To implement this solution we need to have control over the Host entity part of the stack on the *tInterface*, such as the L2CAP protocol layer, in order to be able to create and manage pure L2CAP connections. The Bluetooth module as it is used is not able to perform those tasks. The only option when implementing this solution is to find a Bluetooth module that allows performing the previously referred tasks.

An option to implement the solution is the Bluegiga WT12 Bluetooth 2.1+EDR [24] module, which allows for creating and managing L2CAP connections, as well as some few other types of connections, profiles and general Bluetooth configurations over the UART or USB interface it provides. The proprietary stack running is named iWrap, and allows for a large number of configuration options as well as profiles. This module allows more precise control of the stack parameters, ultimately enabling a deeper control of the connection. Although not an open source stack, a powerful API provides mechanisms to change, while in operation, features such as pairing, device role, and create pure RFCOMM, L2CAP and HCI connections.

Although the solution seemed viable, it was later discovered that with the special hardware required to communicate with Apple products, performance is much lower and never allow for this product's requirements to be fulfilled using Bluetooth Classic. As Apple's Made For iPod (MFi) program documentation is protected under NDA, no further explanation can be provided. At this point, implementing BR/EDR ceases to appear as viable solution, and is discarded.

3.2.3 Bluetooth Low Energy

As Apple's restriction regarding specific hardware to communicate with its devices is only for BR/EDR based physical layers, using Bluetooth Low Energy may allow us to meet the desired requirements. Before Bluetooth 4.2 it would be harder, or even impossible, to fulfill our requirements using Bluetooth LE because the baseband maximum payload size was 27 octets, possibly resulting in crippling overhead and not being able to provide the required throughput for application data. A new feature, packet length extension, is introduced in specification 4.2, making this approach technically appealing as the new packets support payload sizes up to 251 bytes.

Apple's iPad supports Bluetooth 4.2 and although a maximum of up to 251 payload bytes can be negotiated, the actual maximum size in both devices depends on their stack implementation. Some OS's provide for an interface to change this value, while others like Apple manage it internally.

The period for data transmission would also be negotiated to the smaller value possible, enabling for more packets per unit of time.

Various modules and embedded systems in the market are equipped with Bluetooth 4.2, like the Espressif ESP32 module, the Texas Instruments CC26xx based modules as well as Nordic Semiconductors NRF5x solutions.

With available modules and being supported by the iPad without the need for external hardware such as with BR/EDR, this solution appears viable and implementable.

With a strong possibility of BLE v4.2 being a viable solution, two different modules were chosen to be tested, in order to evaluate characteristics such has if they are able to achieve the requirements, ease of programming, accessory features, power consumption and market/industry considerations.

Another factor when considering BLE, is the fact that, with no restraints to the implementation of devices that communicate with iOS over BLE such as the need to use a proprietary protocol, the community and corporate support provided by *e2e* (Engineer-to-Engineer) forums and by technical support platforms is far more accessible, being discussed openly as there is no documentation subject to NDAs.

3.2.4 Bluetooth 5.0

With Bluetooth 5.0 improvements on data transmission layer were introduced. The maximum data rate supported by the LE PHY in this specification is double than in v4.2, allowing for 2 Mbps. With this possibility, data could presumably be transmitted with double the frequency.

This feature will improve performance if supported by both endpoints. Even if Apple's API does not allow for negotiating connection parameters, the data rate would still nearly double what it was, due to the faster transmission of packets.

Bluetooth 5.0 modules are already available for purchase and supported by manufacturers.

Currently the only Apple equipment supporting Bluetooth 5.0 is the iPhone X model. Although the application is not intended to be run on an iPhone, if successfully implemented, this

solution would also allow acquiring knowledge about the most recent Bluetooth specification. When eventually an iPad model that supports the new version of Bluetooth is released, the company would be ready to update its technology.

Testing of this solution would be a great opportunity to use the latest technology, also a favorable factor for the product.

3.2.5 WiFi

This technology would most likely provide an answer to the problem, as the physical layer supports data rate much higher than required. However, a direct connection between the devices in ad-hoc mode would be required in order to maintain product usability. With the need for a router, a series of usability problems would arise, such as the tablet not being able to access the Internet if the *tInterface* were to be an AP, or having to configure both devices to connect to a local router in every place the product were to be used. Such approach is not impossible, but not viable in terms of usability and therefore, marketability. - Some concerns for this solution arise from health system regulations regarding data protection and WiFi usage. Different laws and regulations exist depending on countries and/or regions, and an ever increasing concern for security in medical devices exist[25]. Power consumption is also normally significantly higher within devices that use WiFi technology, making it a less attractive solution than its counterparts.

3.3 Summary

This chapter discussed the problem as formulated by Adapttech, in which the existing product is not able to meet the throughput requirements. This is surely related either with the receiving end of the application messages (iPad) or the BM78 Bluetooth module. As the information regarding the iAP protocol is under an NDA and the implementation of the Bluetooth stack running in the BM78 is also private IP, a deeper understanding of the Bluetooth specification is needed in order to be able to find, understand and provide a solution for the described problem.

Various solutions have been considered and, although every one of them is technically achievable, some were discarded as they are not implementable in the corporate environment due to other, non technical, restrictions. Examples of these restrictions are usability considerations such as how a user would interact with a device, how the connections with the system are created and the configurations required in order to operate the device. Technically, the main constraints to consider were the use of an iPad and Bluetooth communication.

For the sake of solving the problem at hand, we needed to consider both effective and application throughput. For this purpose, we used a protocol sniffer for throughput measurement, as it can provide an observer's perspective, not interfering with any of the process components and allowing for a more detailed view of connection requests, parameter negotiation and data message segmentation.

This way we expect to be able to pinpoint the effective reasons for the BM78's behaviour and also to know how to create and implement a Bluetooth device that meets the requirements, focusing on BR/EDR and BLE.

Chapter 4

Bluetooth Technology

As Bluetooth is the existing solution found in the *tInterface*, a detailed analysis of the several specifications will be performed, in order to understand how the device communicates and to be able improve performance, satisfying requirements and providing an in-depth look of Bluetooth Technology.

4.1 Bluetooth profiles

Bluetooth profiles are one of the key points in this technology. While different parts of its stack have features which make interoperability a possibility, with profiles that possibility is materialized. Profiles are cross-layer to the Bluetooth stack (Fig 4.1), although most of the time they are said to be at the top of the stack, just below the application. They provide the stack layers with predetermined interactions, creating several different standard behaviours for Bluetooth devices. The following image shows an abstract relation between the stack layer and profiles. There are profiles serving a wide range functions, from medical devices to simple thermometers.

4.1.1 Serial Port Profile

This profile was designed in order to support legacy devices using virtual COM ports, providing a simple interface for legacy applications[26].

> "The applications on both sides are typically legacy applications, able and wanting to communicate over a serial cable (which in this case is emulated)"

The Serial Port Profile (SPP), provides applications a virtual serial port, completely transparent to the operating system. SPP states a need to provide reliable communications, such as the ones in the RS-232 protocol so, with this in mind all procedures and characteristics of this profile are made in order to provide the best reliability possible. This profile uses the RFCOMM protocol in order to establish a Data Link Connection (DLC), from a initiating device known as DevA, to a remote Bluetooth device, DevB. SPP also dictates configuration options and parameters for

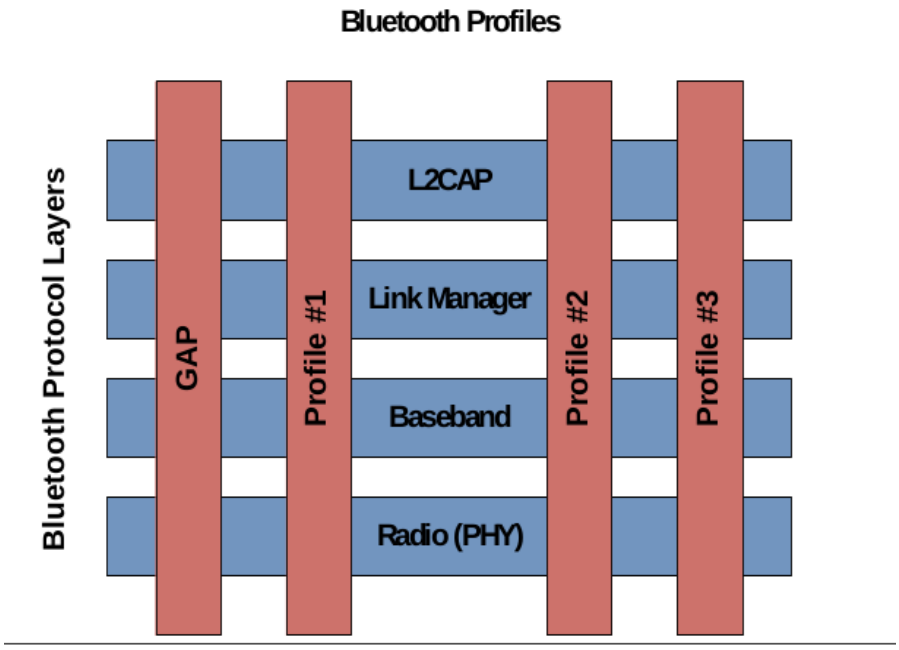

Figure 4.1: Bluetooth profiles [2]

lower layers, such as L2CAP. An option is also provided for SPP to emulate RS-232's flow control signals, RTS/CTS, dependent on API.

The connection establishment procedure for SPP is described shortly next:

- Discover Remote RFCOMM Server channel.
- Execute authentication and encryption procedures (optional)
- Create L2CAP connection-oriented channel to remote entity.
- Start an RFCOMM session and establish a Data Link Connection.

The protocol stack and device interaction can be seen in Fig. 4.2.

Just below the Serial Port Emulation entity is the RFCOMM protocol,based of the GSM TS 07.10 specification, used for terminal-modem communications in Global System for Mobile Communications (GSM).

L2CAP is a transport protocol used in Bluetooth communications and, as such, SPP imposes L2CAP connection parameters such as it's Maximum Transmission Unit, Flush Timeout and recommendations for flow and error control. The flush timeout parameter can be seen as how much time a baseband packet can be waiting on a buffer to be re-transmitted. In order to provide a reliable link, L2CAP is configured with an infinite flush timeout, therefore disabling it from discarding packets.

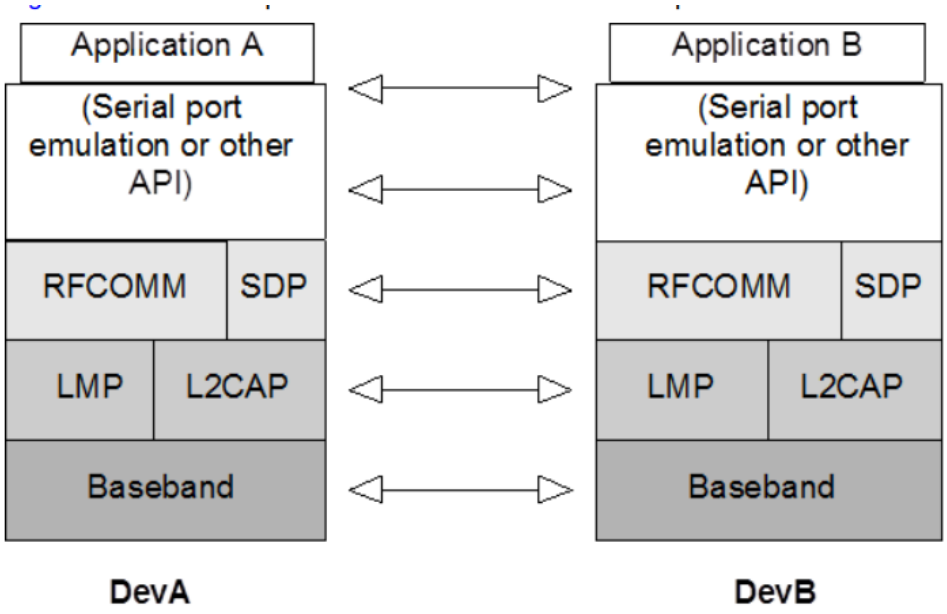

Figure 4.2: Protocol Stack used by the Serial Port Profile

4.1.2 Health Device Profile

Bluetooth Classic has an accessory profile, specialized for use with medical devices, in order to provide easy connectivity and compatibility. Health devices must follow this specification in order to be qualified for Bluetooth Healthcare and Fitness. There are two main abstract roles defined in this specifications, sources and sinks, with each device being either a sink, source, or both. Sources transmit information, whilst sinks are receivers of information. Several possible configurations are also defined, in order to cover most use cases.

In order to support those topologies, HDP uses the Multi Channel Adaptation Protocol in order to create an abstraction from the L2CAP layer. Data is exchanged using the IEEE 11073 Health Informatics - Medical / health device communication standards. The complete stack can be seen in Fig. 4.4 as well as how two devices communicate with another.

4.2 Bluetooth Networks

For better understanding, we will briefly discuss network topologies before an in-depth analysis of the Bluetooth stack, to provide readers with a basic understanding of the theory of operation within Bluetooth Networks. With Bluetooth Classic, two topologies were introduced, piconet and scatternet. The main difference is that a piconet is a network containing only one BT device operating as the master in the connection. A device is said to be in a scatternet if it is active in more than one piconet, consequently having more than one master node.

Below are two illustrations representing the possible network topologies involved, for BR/EDR and Bluetooth LE.

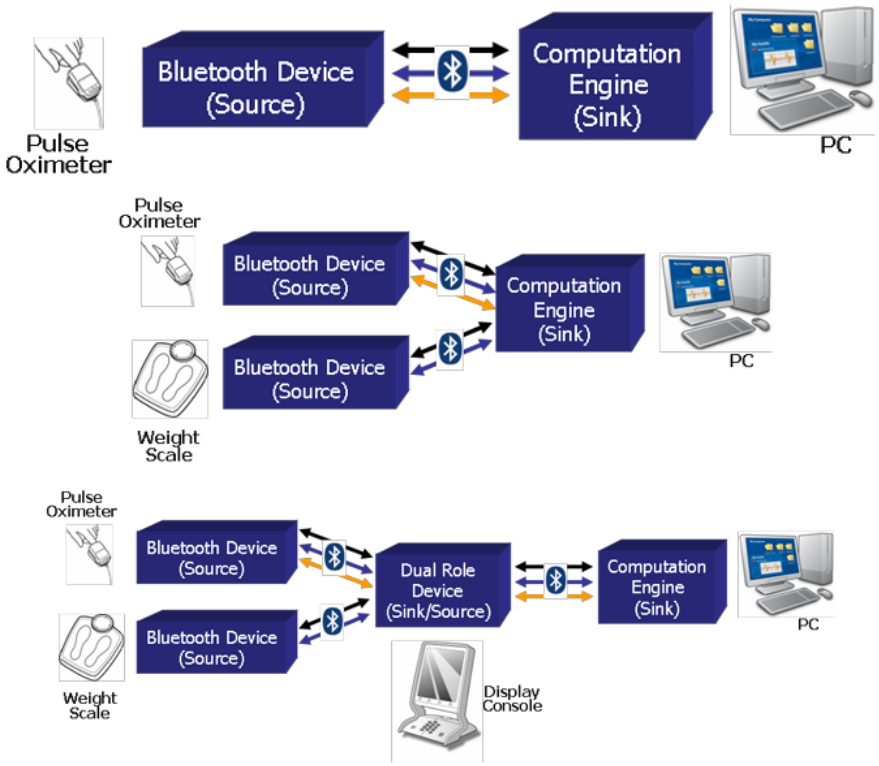

Figure 4.3: HDP Network Topologies [2]

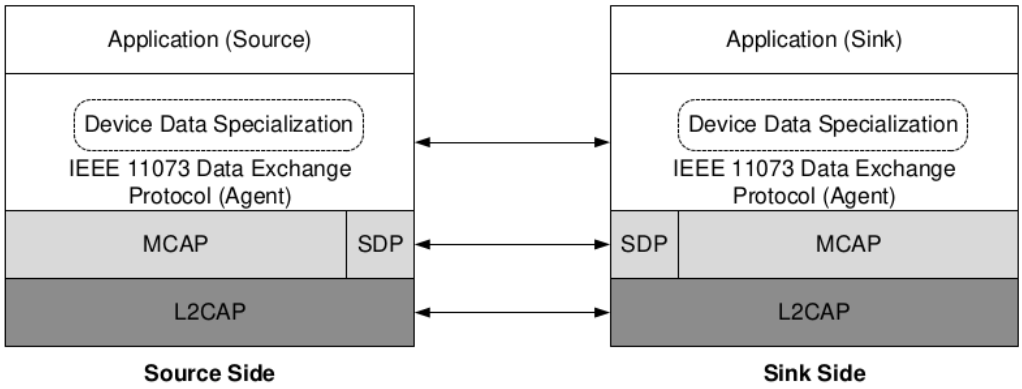

Figure 4.4: HDP Stack [2]

In a BR/EDR piconet, all slaves are tuned into the master channel and frequency hop. A piconet can support up to 7 slaves, and allow for unicast and broadcasting communications. In LE, slaves within the same piconet use different channels to communicate with the master, each slave having its own channel. The maximum number of slaves is directly related to a part of the packet header, LT ADDR, which is 3 bits long, making it possible to address the 7 slaves and one extra address for broadcast.

In LE device networks scatternets are not defined, although a device can be both executing a central and peripheral role, being a master and a slave at the same time in different piconets. In essence, each group of slaves connected to a master forms a piconet, and a master can also be

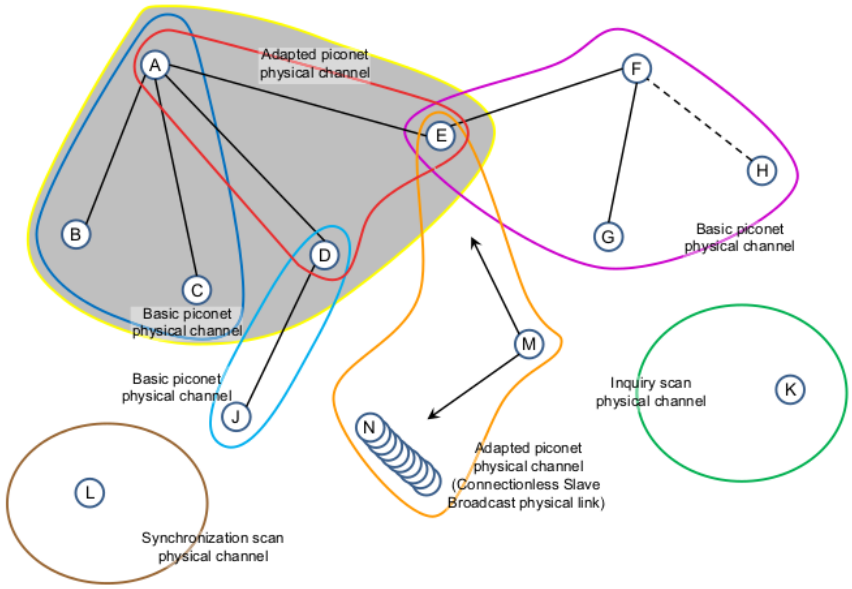

Figure 4.5: BR/EDR Network Topologies [2]

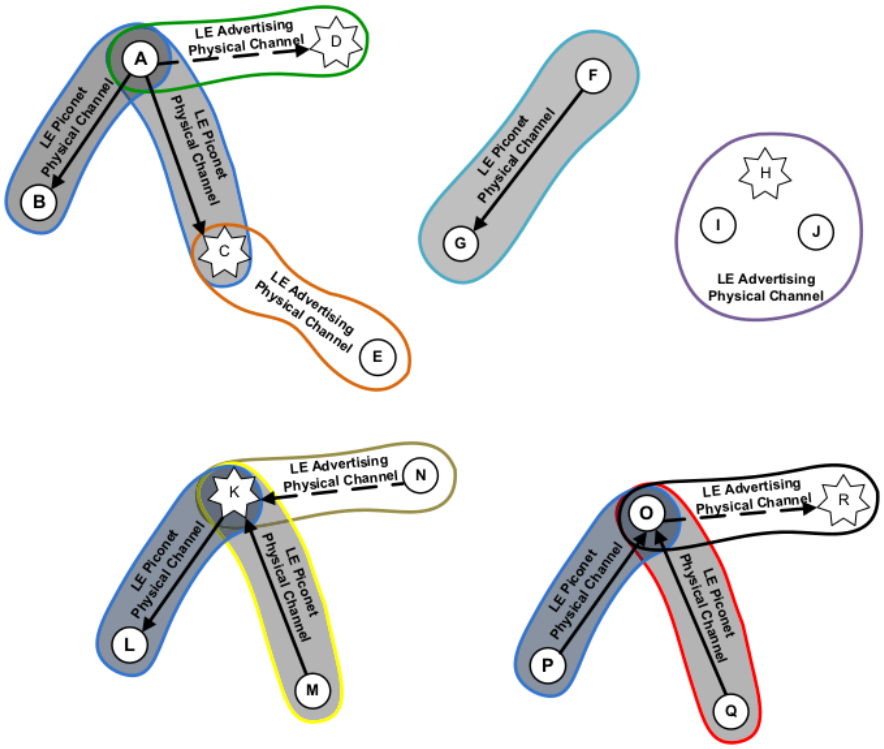

Figure 4.6: LE Network Topologies

a slave to another piconet or, be a master to two different piconets. These topologies create the possibility for a device to be actively communicating with one network, whilst being discoverable and pairable to another network. The connections represented in the figures above are defined in the Link Layer of the Bluetooth Stack. To create them, Bluetooth Specification supports several device operations like scanning for other devices, pairing and connecting with scanned devices,

among others.

4.2.1 Connection Modes

In order to support different features and enable networks to operate under different conditions and with several slaves, the Bluetooth specification defines several connection states.

4.2.1.1 BR/EDR Connection Modes

Connections within BR/EDR networks can be in one of several states:

- Active Mode - In this mode, connections are actively engaged in data transmission
- Sniff Mode - Sniff mode implies that a Bluetooth device will listen to the medium with a lower frequency, lowering it's duty cycle.
- Hold Mode - This other power saving mode enables connections to be held for a specific time frame.
- Park Mode - When slaves are parked, they yield the ability to transmit data, only listening for broadcasts and synchronization packets.

4.2.1.2 BLE Connection Modes

- Connected Mode - In this mode, connections are actively engaged in data transmission, where both devices are directly connected, sharing a physical data channel.
- Advertising mode - In this mode, data can be transmitted through advertisement channels and can be received by other devices. It can be used to transmit information, or to scan, retrieve information and connect to other BLE devices.

4.3 Bluetooth Stack Overview

Bluetooth features two main architectures, BR/EDR and LE, and these differ in numerous aspects. Fundamental characteristics will be presented in this next section, having a ever more in-depth look on how both core systems components relate with another, and how operations are performed. The stack provides mechanisms and functions to support operations required by the profiles functionalities.

Referring to Figure 4.7, it can be seen that generally an application interacts with a profile, and this profile executes different procedures and controlling lower layers in order to provide a connection with other devices and a configured channel for data flow. Another observable fact, is that the LE and BR/EDR devices employ different stacks. The RFCOMM protocol, used by many profiles, is not used in Low Energy devices. Data in LE devices is normally exchanged on a client/server configuration using the General Attribute Table, where different services are composed of different characteristics and attributes that are ultimately store and present data.

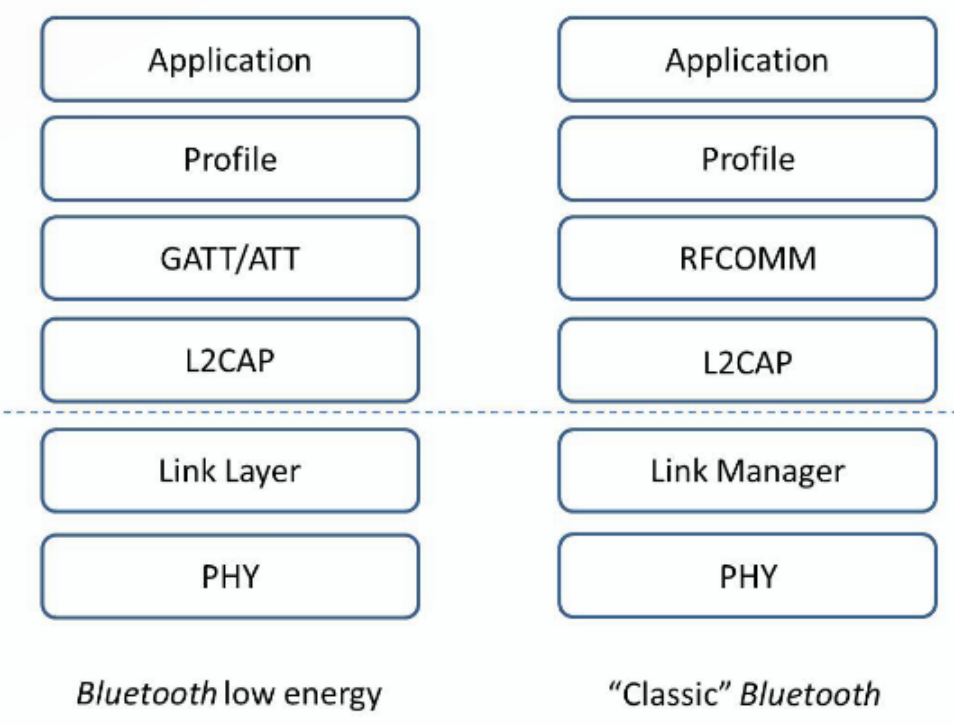

Figure 4.7: BR/EDR and LE stack comparison [3]

4.3.1 RFCOMM

RFCOMM or Radio Frequency Communication is a protocol designed to provide an abstraction to be able to virtualize Bluetooth services as a serial port. The protocol is responsible for delivering to an application an interface to be able to read and write data though a virtual serial port, also dealing with it's signaling. The reference model for RFCOMM is show in Fig. 4.8:

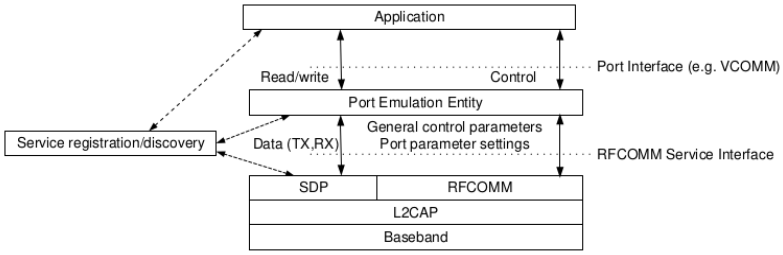

Figure 4.8: RFCOMM model [4]

RFCOMM, adapted from TS 07.10, has two main types of frames, which are used for communications, frames that are intended to carry signaling and control information, and frames intended to carry data. A connection between to RFCOMM entities, is called a Data Link Connection (DLC).

Different types of frames can be sent over a DLC, with different intended uses such as control or data services. These frames are:

- SABM - Set Asynchrounous Balanced Mode command
- UA - Unnumbered Acknowledgement response
- DM - Disconnected Mode response
- DISC - Disconnect Command
- UIH - Unnumbered information with header check command and response.

Several parameters are defined in an RFCOMM entity as shown in Fig. 4.9

System Parameter	Value
Maximum Frame Size *(N1)*	Default: 127 (negotiable range 23 – 32767)
Acknowledgement Timer *(T1)*	10-60 seconds. Recommended value: 20 seconds. See also note below.
Response Timer for Multiplexer Control Channel *(T2)*	10-60 seconds. Recommended value: 20 seconds. See also note below.

Figure 4.9: RFCOMM System parameters [4]

Regarding reliability, RFCOMM does not provide any direct mechanisms to assure it and as such for each timeout, a connection should be closed and re-established. The defined parameters for Data Link Connection parameter negotiation can be seen in Fig. 4.10

- I1-I4 shall be set to 0. (Meaning: use UIH frames.)
- T1-T8 shall be set to 0. (Meaning: acknowledgment timer T1, which is not negotiable in RFCOMM.)
- NA1-NA8 shall be set to 0. (Meaning: number of retransmissions N2; always 0 for RFCOMM.)

Figure 4.10: RFCOMM DLC parameters [4]

> "(...)RFCOMM must require L2CAP to provide channels with maximum reliability, to ensure that all frames are delivered in order, and without duplicates. Should an L2CAP channel fail to provide this, RFCOMM expects a link loss notification (..)"

This protocol defines two devices, DevA and DevB, supporting several multiplexed serial connections. Emulation of the serial port requires modem control functions, port and parameter negotiations as well as a method for information transmission. An example of the frames used by RFCOMM can be seen in Fig 4.11. This protocol enables a set of features such as flow control, baud rate selection and methodologies to set up and extinguish connections. Different types of frames are defined in order to allow for modem communications, such as creating Data Link Connections (DLC). RFCOMM is defined as a reliable stream of data, as such, it requires the underlying L2CAP channel to be configured with a infinite flush timeout. Also, as the link is supposed to be reliable, the specification states:

Address	Control	Length Indicator	Information	FCS
1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet

Figure 4.11: RFCOMM frame [4]

"If an L2CAP link loss notification is received, the local RFCOMM entity is responsible for sending a connection loss notification to the port emulation/ proxy entity for each active DLC. Then all resources associated with the RFCOMM session should be freed."

4.3.2 Bluetooth Core Systems Overview

Every Bluetooth System is composed by two main entities, the Host and the Primary Controller. According to the specification defined by the Bluetooth SIG, a Controller is defined as a logic unit, responsible for the lower stack layers, such has the baseband, physical layer and a part of the Host Controller Interface (HCI). The other logic entity defined is the Host, which implements the higher layer protocols and profiles like the Service Discovery Protocol (SDP), Attribute Protocol (GATT/ATT) and L2CAP. The interface between both entities is the Host Controller Interface (HCI) and it specifies a set of commands and parameters supported by both entities, to transmit data between both parties. Commands are sent from the host to the controller, and events are reported from the controller back to the host platform. Both parties may be implemented in the same device, such as in many Bluetooth modules that provide a transparent UART service, or, be physically separate such as the case of Bluetooth dongles that connect via USB to a PC.

Two types of controllers are defined, primary and secondary controllers (Fig. 4.12). Primary controllers are BR/EDR and LE cores, and secondary controllers are defined as AMP controllers. AMP controllers were only introduced in specification 3.0, giving the possibility to use a different physical layer for transport of data, using the same higher layer protocols. This was introduced into the Bluetooth specification in order to support different radio standards, such as the 802.11, taking advantage of the different characteristics of various physical layers such as increased bandwidth. Using the defined logic entities, controllers and hosts, a Bluetooth radio can contain one or more of those, in different configurations as the following images illustrate.

A global overview of the Bluetooth Core System is presented in Fig. 4.13, in which all three types of Controllers are present, although that does not mean every implementation has to be composed of the three.

Directly above the physical channel and link layers are the logical links, logical transports and L2CAP layers. Each of these layers play an important role, making Bluetooth a versatile protocol

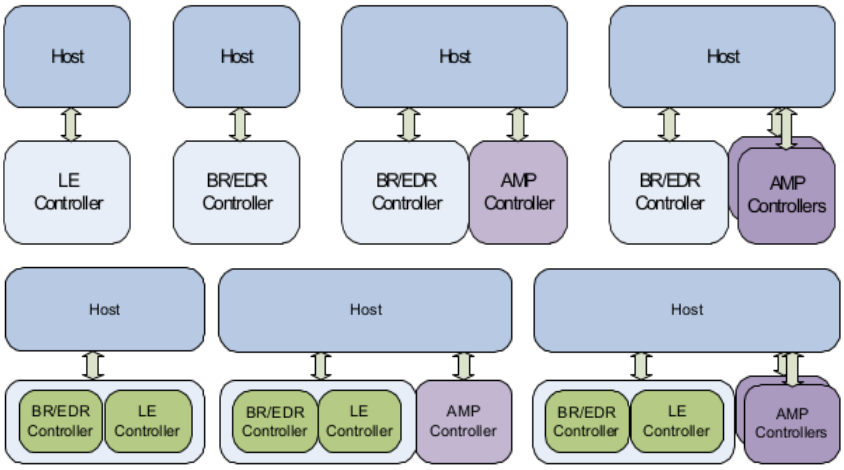

Figure 4.12: Single/Dual Mode Configurations [2]

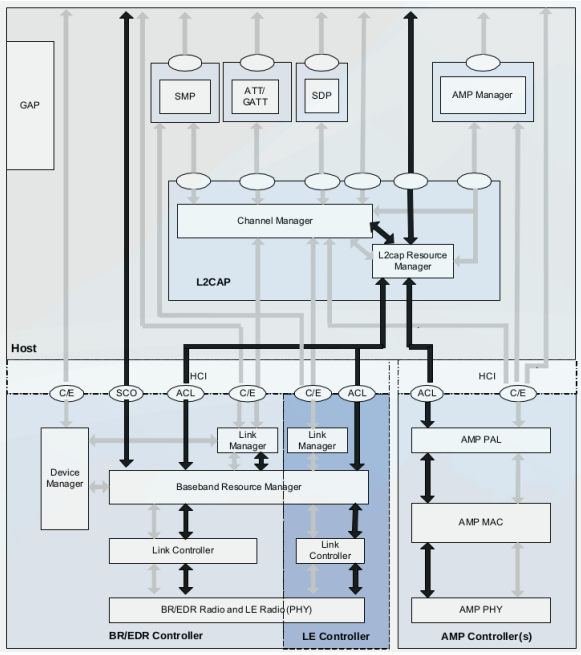

Figure 4.13: Bluetooth Core System [2]

supporting data with different timing characteristics and providing mechanisms to convey both user and control data to the appropriate entity.

The abstract layering used for data transportation is illustrated in Fig. 4.14 and Fig. 4.15.

In Fig. 4.14, several layers can be identified, each subdivided into possible combinations of transports, channels and links. A more detailed illustration is shown in the Fig. 4.15, elucidating which entities are involved in each layer. Any kind of data sent via Bluetooth is sent from the upper layers to the lower ones, being transmitted and ,after being delivered, are delivered back to the application on the other endpoint device(s).

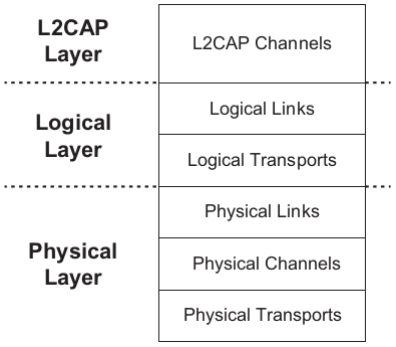

Figure 4.14: Detailed Bluetooth Data Transport Structure [2]

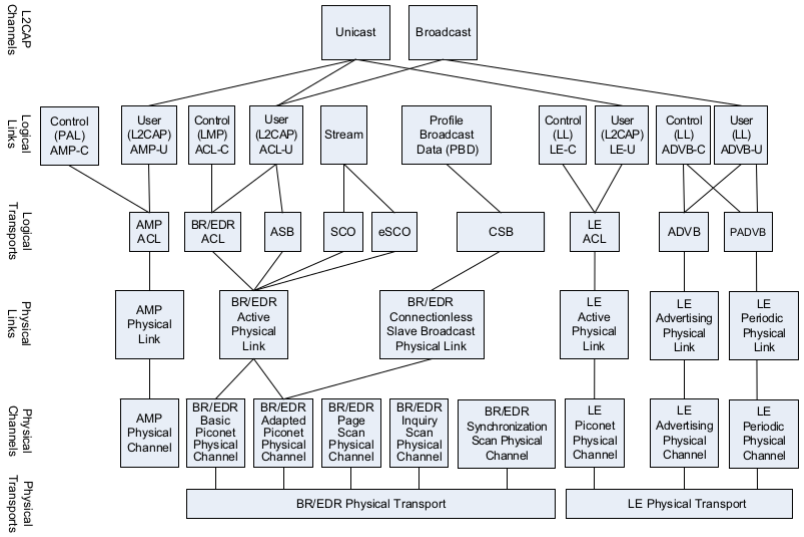

Figure 4.15: Bluetooth Data Transport Architecture [2]

4.4 Logic Link Control Adaptation Protocol

Although every one of the protocols appending the Bluetooth specification are important, L2CAP is essential regarding application data transmission. Offering packet multiplexing, quality of service options and segmentation and reassembly, this protocol is imperative in most cases involving transmission of data for applications. Providing these services, L2CAP creates an important layer of abstraction in order to facilitate higher protocol functions, simplifying some necessary complex and critical tasks mandatory for reliable communication. L2CAP's interaction with other upper and lower layers is also defined, using a scheme used based on an asynchronous request-response technique.

4.4.1 Modes of Operation

As different types of traffic are required by applications,L2CAP features several modes of operation:

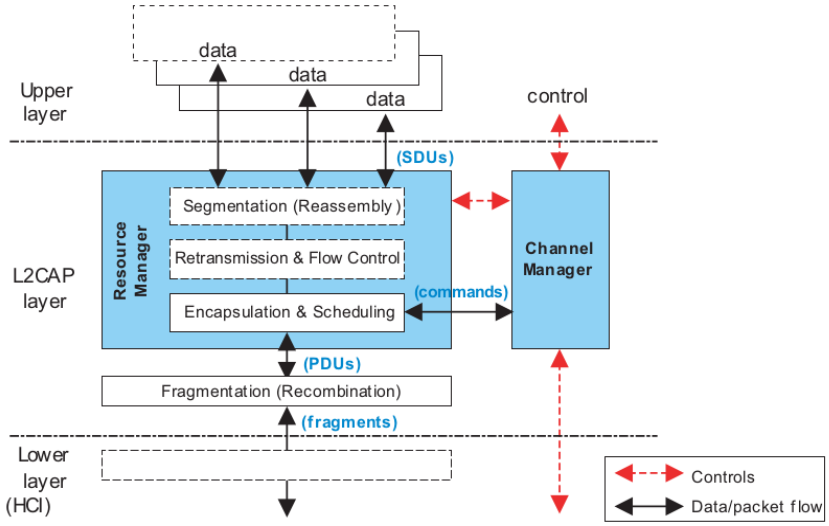

Figure 4.16: L2CAP architecture [2]

- **4.4.1.1 Basic L2CAP Mode**

In Basic L2CAP Mode there are two types of frame used to support connection-oriented and connectionless traffic, respectively known as G-Frame and B-Frame. These can be seen in Fig. 4.17.

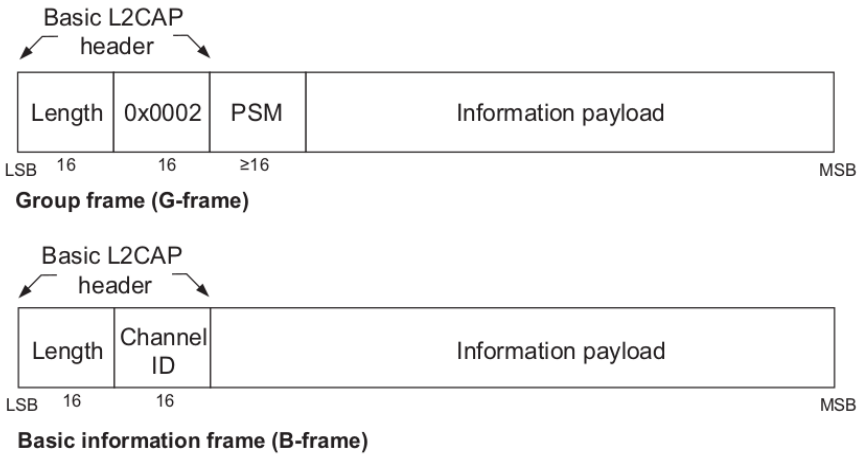

Figure 4.17: Basic Information and Group Frames [2]

- Flow Control Mode, Retransmission Mode, Enhanced Retransmission Mode and Streaming Mode In these 3 operational modes, Supervisory and Information Frames are used in order to permit flow control, retransmission and streaming. S-frames are used to supervise the connection, being used according the defined operational mode and performing acknowledgement and retransmission requests.

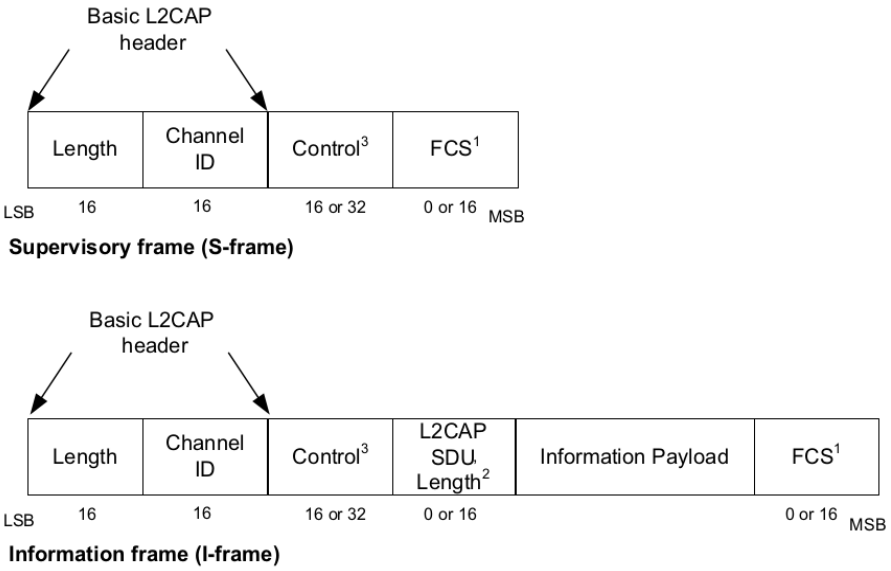

Figure 4.18: L2CAP Information and Control frames [2]

- LE Credit Based Flow Control Mode This mode of operation is only used with LE systems resorting to LE information frames (LE-frame) 4.19.

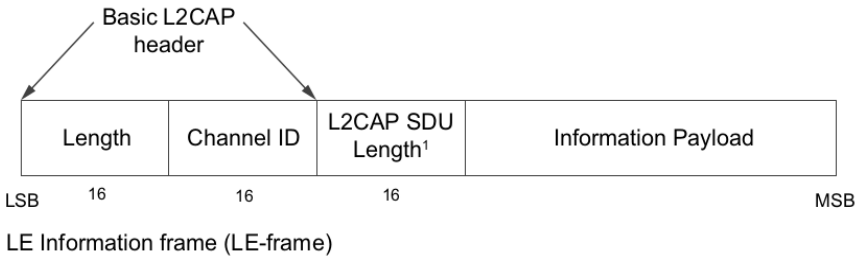

Figure 4.19: LE Information Frame [2]

*

4.4.2 Fragmentation and Reassembly

After receiving SDUs, L2CAP normally delivers its PDU to the Host Controller Interface (HCI), with is responsible for communication with lower layers in the present in the Bluetooth Controller. As information is tranfered from the L2CAP layer to the physical radio interface, it is fragmented and reassembled by various controllers and protocols. An illustration of this process can be seen in Fig. 4.20.

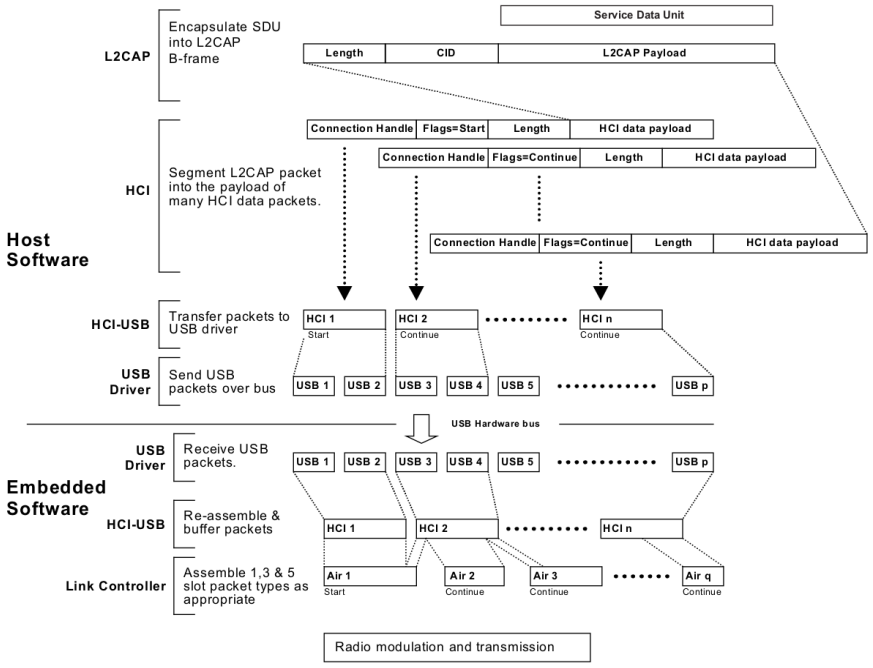

Figure 4.20: L2CAP and Baseband information flow [2]

4.5 Generic Access Profile

The Generic Access Profile (GAP) is a mandatory profile that must be present in every Bluetooth device, as it enables required procedures devices have to perform in order to establish and maintain connections. GAP defines device roles and procedures and modes for discoverability, connections and security. A hierarchical view is given in the following image.

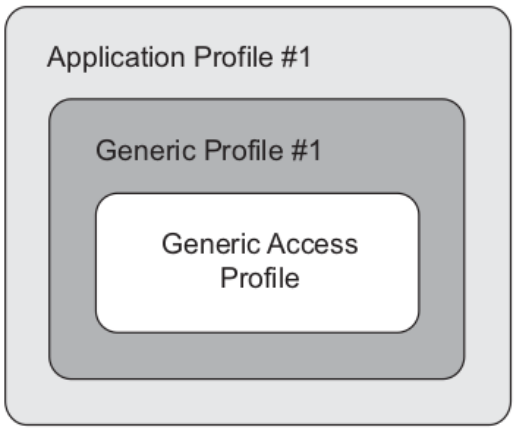

Figure 4.21: Generic Access Profile Hierarchy [2]

In BR/EDR GAP defines two roles for devices, the initiator of the connection known as A-party and acceptor, B-party.

For LE systems, GAP defines 4 roles, for use in more specific cases, with its own characteristics. The Broadcaster role is relevant for use in applications that only transmit data. On the

other side, the Observer role acts as a listener, normally used for applications only receiving data. Peripheral role enables a device to be able to support a single connection, giving the possibility to make cheaper, smaller and lower consuming devices. The Central role is normally performed by devices such as a laptop, in this role devices support several connections with peripherals and are the ones responsible for initiating connections. Advertisement, device discoverability and related procedures are also defined in the GAP specification, effectively creating an entry point to a device, also handling scanning and connection creation.

The GAP specification also defines a user interface level, where the representation of parameters on a UI (User Interface) shall be displayed, such as the Bluetooth device address, friendly name and security procedures at a higher level. As figure 4.22 shows, GAP interacts with a big part of the stack, as functions performed by it are fundamental for Bluetooth device to operate.

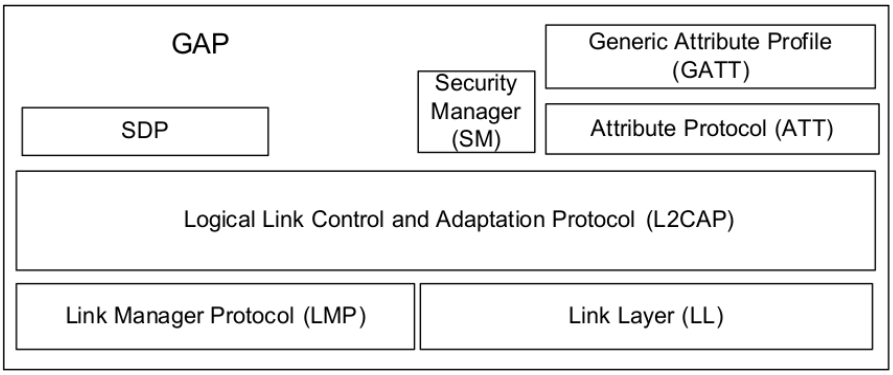

Figure 4.22: Generic Access Profile Stack [2]

4.6 Generic Attribute Profile

This profile uses the Attribute Protocol (ATT) in order to transfer data, defining the type of data used and its formatting. GATT profiles are composed of one or more services, in turn composed of characteristics. Characteristics are a set of one or more values and their properties such as how the value is displayed and accessed (Fig. 4.23).

Each GATT transaction relies on lower layer protocols to transfer information and directly below is the Attribute Protocol (ATT) 4.24, responsible for the transmission of data using a request/response scheme.

GATT data is transferred using a client/server architecture, where usually the client sends requests for data and the server replies with the corresponding data or an error code. Several procedures are defined, in order to discover, read and write data. There are various types of procedures for writing and reading characteristics values, in order to fit different use cases:

- Read Characteristic Value - This is the simplest procedure to read a characteristic value, based on the attribute handle, making a request and receiving the subsequent response.

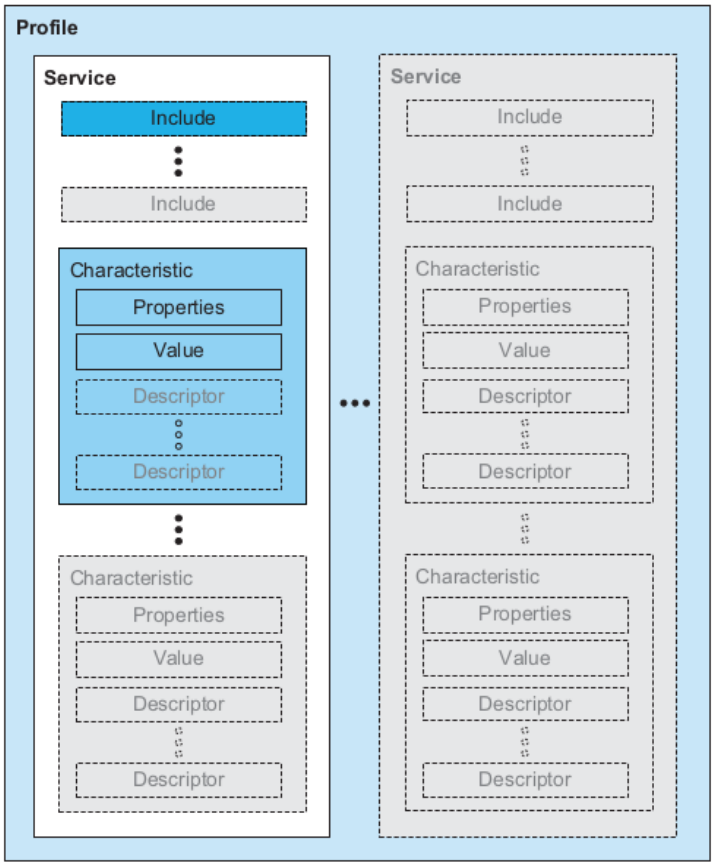

Figure 4.23: Generic Attribute Profile Hierarchy [2]

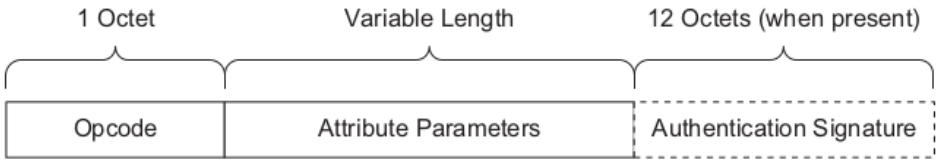

Figure 4.24: ATT PDU [2]

- Read Using Characteristic UUID - Reading based on the Universally Unique Identifier.
- Read Long Characteristic Values - This sub-procedure enables reading of a characteristic value longer then the defined ATT Maximum Transmit Unit.
- Read Multiple Characteristic Values - This sub-procedure is used to facilitate the reading of several characteristic values using only one request
- Write Without Response - This sup-procedure writes a value to a characteristic's value, without the need for acknowledgement .

- Signed Write Without Response - Like the previous write sub-procedure it writes a characteristic value, but the ATT PDU contains a field with a signature in order to enable authentication of the sending device.
- Write Characteristic Value - Using this write method implies a response from the server thus serving as an acknowledge message.
- Write Long Characteristic Write - Has the same function as the previous procedure, but enables the write of a value longer than the ATT MTU.
- Characteristic Reliable Write - A reliable write is different as it is done in two separate phases, first transferring the data and checking it's validity and in the second phase the actual value of the characteristic is changed.
- Notification - This sub-procedure is used for a GATT server to update a characteristic value in a client. This type of procedure does no imply any type of acknowledgment.
- Indication - An indication is similar to a notification, but requires acknowledgment.

4.6.1 L2CAP interoperability

GATT also defines a series of requirements for lower layers, more specifically L2CAP, in order to assure implementation compatibility between devices. ATT transfers are done using an L2CAP channel, referred to as the ATT bearer. This bearer is different for use in BR/EDR and LE PHYS, each with it's own default parameters and configurations.

The predetermined L2CAP packet used in this channel, for both BR/EDR and LE, is the B-Frame, also known as Basic Information Frame.

A standard BR/EDR ATT Bearer is configured for reliable traffic with a flow specification used is best effort, using a dynamic channel ID and a fixed Protocol Service Multiplexer (PSM). In Bluetooth LE, requirements are similar to Bluetooth Classic, using a fixed channel ID, transmitting data with reliability with the retransmission and flow control mode set to Basic L2CAP mode.

Default parameters for an LE ATT Bearer can be seen in Fig. 4.25.

Parameter	Value
MTU	23
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Basic Mode

Figure 4.25: Default LE ATT Bearer parameters [2]

4.7 Host Controller Interface

In this section of the specification, every possible interaction between the host and the controller is defined. Three hardware interfaces can be used, USB, UART and Secure Digital(SDIO). The controller typically receives commands from the host and sends events back to the controller. A set of commands and responses are defined by the Host Controller Interface, supporting communication between the Bluetooth Host and Controller.

4.8 Bluetooth Basic Rate / Enhanced Controller

The BR/EDR defines a set of protocols and entities that, together with the physical radio specified, enable devices to operate at 1, 2 and 3 mb/s. It defines a data transport architecture, in order to provide communication channels for different types of traffic, and protocols in order to enable physical packet fragmentation reassembly and support different types of networks.

4.8.1 Bluetooth Baseband, Physical Channels, Links and Transports

The Bluetooth Link Controller specification dictates the operating methodology of the BR/EDR baseband, as it defines a set of packets and timing conditions for each type of physical channel. The Bluetooth Link Controller is also directly responsible for managing timing slots necessary for the communication to occur. Different types of packets are defined for the baseband layer, these answer specific needs of different physical transports. Packets differ in function and can occupy from one to five time slots, depending on packet type, each slot having a duration of 625μs .

The Link Controller manages connections and procedures, such as the entry and exit of piconets and scatternets. The controller can be described as a state diagram. There is a primary state as well as several secondary, temporary states. Transition between states are either started because of internal controller events or commands from the link manager. Fig. 4.26 shows the state diagram.

Physical channels are the bare core of Bluetooth communication, every transaction occurs in one of these channels. A channel is characterized as a set of frequencies, temporal parameters and spatial considerations. In other words, for a device to be able to communicate with another, they must be tuned exactly at the same time to the same frequency. It is assumed that different devices on various networks are probable to operate within the same space, and to mitigate collisions each channel is identified with an access code. There are several channels in order to fulfill different needs, like device discovery, network synchronization and data transmission on established connections. As only one channel can be used at a time, time division is used to imply parallelism to operations, however different channels may be able to collide in the same frequency.

An overview of the various links and transports in given in Fig. 4.27.

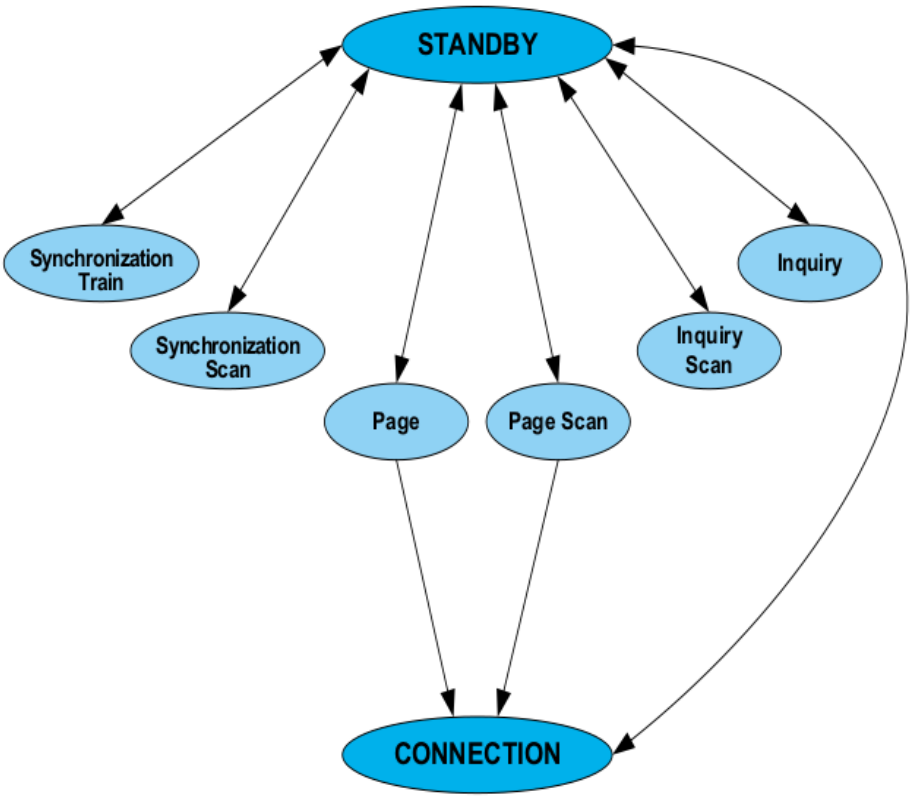

Figure 4.26: BR/EDR Link Controller States [2]

4.8.1.1 Logical Links and Transports

In BR/EDR there are 4 types of logical transports, and each has it's own characteristics. There are two logical transport links responsible for synchronous data, the Synchronous Connection Oriented (SCO) and Extended Synchronous Connection Oriented (eSCO). Those logical transports are mostly used in use cases such as audio communications, which transmit data on a specific and previously established data rate. The other two are divided into Asynchronous Connection-Oriented (ACL) and Active Slave Broadcast (ASB).

Logical Links provide an abstraction, separating user and control data. I.e., the ACL-U Logical Link is used for the to deliver user data to upper layer protocols such as RFCOMM, while the ACL-C Logical Link is responsible for delivering control data to the L2CAP resource manager.

4.8.1.2 Addressing

Each Bluetooth device has a unique 48 bit address, defined as a EUI-48 (Extended Unique Identifier). The device address for Bluetooth devices is not, as other protocols/specifications, just a unique reference, as it is an important property which is used for determining how the connection will be set up, such as channel access code and critical operations such as data whitening. The address itself is divided in to three parts, the LAP (Lower Address Part), UAP (Upper Address

Logical transport	Links supported	Supported by	Bearer	Overview
Asynchronous Connection-Oriented (ACL)	Control (LMP) ACL-C or (PAL) AMP-C User (L2CAP) ACL-U or AMP-U	BR/EDR active physical link, BR/EDR basic or adapted piconet physical channel, AMP physical link, AMP physical channel	BR/EDR, AMP	Reliable or time-bounded, bi-directional, point-to-point
Synchronous Connection-Oriented (SCO)	Stream (unframed) SCO-S	BR/EDR active physical link, BR/EDR basic or adapted piconet physical channel	BR/EDR	Bi-directional, symmetric, point-to-point, AV channels. Used for 64Kb/s constant rate data.
Extended Synchronous Connection-Oriented (eSCO)	Stream (unframed) eSCO-S	BR/EDR active physical link, BR/EDR basic or adapted piconet physical channel	BR/EDR	Bi-directional, symmetric or asymmetric, point-to-point, general regular data, limited retransmission. Used for constant rate data synchronized to the master Bluetooth clock.
Active Slave Broadcast (ASB)	Control (LMP) ASB-C User (L2CAP) ASB-U	BR/EDR active physical link, basic or adapted physical channel	BR/EDR	Unreliable, uni-directional broadcast to any devices synchronized with the physical channel. Used for broadcast L2CAP groups and certain LMP messages.
Connectionless Slave Broadcast (CSB)	Profile Broadcast Data (PBD)	Connectionless Slave Broadcast physical link, BR/EDR adapted piconet physical channel	BR/EDR	Unreliable, unidirectional, point-to-multipoint, periodic transmissions to zero or more devices.
LE asynchronous connection (LE ACL	Control (LL) LE-C, User (L2CAP) LE-U	LE active physical link, LE piconet physical channel	LE	Reliable, bi-directional, point-to-point.
LE Advertising Broadcast (ADVB)	Control (LL) ADVB-C, User (LL) ADVB-U	LE advertising physical link, LE advertising physical channel	LE	Unreliable, uni-directional broadcast to all devices in a given area or directed to one recipient. Used to carry data and Link Layer signaling between unconnected devices.
LE Periodic Advertising Broadcast (PADVB)	Control (LL) ADVB-C, User (LL) ADVB-U	LE periodic physical link, LE periodic physical channel	LE	Unreliable, periodic, unidirectional broadcast to all devices in a given area.

Figure 4.27: Logical Links and Transports Chart [2]

Part) and the NAP (Non-important Address Part), each playing an important role, except for the NAP.

4.8.1.3 Physical Channels

In Classic Bluetooth, the most basic building block for communication is a physical channel, defined as a combination of slot timing, access code, encoding and hopping sequence. In order to consider two device as connected, they need to have these parameters successfully negotiated. As the hopping sequence is pseudo-random, access codes for a channel are needed in case two different piconets accidentally tune into the same frequency at the same time, enabling for a packet from a different piconet to be discarded. Frequency hopping is used in order to reduce EMI, as Bluetooth devices operate in the ISM spectrum.

In BR/EDR there are 5 physical channels, these are:

- Basic Piconet Channel - This is used for data transmission between devices with established connections. Every time slot has a specific frequency, with hops happening between consecutive time slots.

- Adapted Piconet Channel - Being similar to the Basic Piconet Channel, this channel uses another sequence relating to frequency and time slots. In the Adapted Piconet Channel two consecutive time slots generally have the same frequency. This channel supports AFH, being able to use a reduced channel map.

- Inquiry Scan Channel - This channel is used for device discovery, as it awaits inquiry request it browses through every possible inquiry frequency sending a request on each one, and upon a request a response is sent. The channel is also identified by one of two types of access codes, either a general inquiry access code or a limited access code.

- Page Scan Channel - Is used to initiate a connection from one device to another. The device initiating the process will send page requests to the target device, hopping frequencies in order to find the other devices hopping pattern and frequency.

- Synchronization scan Channel - This channel is used for providing clock synchronization to devices during the page process .

Time division multiplexing is critical in order to allow devices to perform actions simultaneously, from a user's point of view, like being connected and scanning devices at the same time.

4.8.1.4 Piconet Clock

The main clock in any Bluetooth device is the Piconet Clock, used as reference for Link Layer scheduling, and practically every baseband function. An offset update is periodically required, since oscillators from different devices will naturally diverge. The Piconet Clock is defined and broadcasted by the piconet Master, normally the initiator device in the connection.

4.8.1.5 Transmission Timing

As previously stated, communications is made using a TDD (Time-Division Duplex) technique, and each slot defined is 625us. There are two different types of slots, master-slave and slave-master, and these are distributed temporally in a that collisions between master and slaves will never occur, as even slots are reserved for master transmissions and odd slots are for slave-master transmissions. A polling scheme is used by the master in order to control slaves in a piconet, so each slave is only able to transmit data if it receives a packet on the previous master-slave time slot.

At the lower physical level, data is transmitted using a packet based communication technique. A packet normally uses only one time slot, but this can be extended, giving the possibility to transmit longer packets, using several time slots at a time. Full duplex communication is supported due to the TDD scheme used. Data is sent making use of physical links, a layer that sits directly above the physical channel, which provides mechanisms to convey information correctly resorting to some level of abstraction. A physical link is used to transport information from different logical links using a multiplexing technique assigning different slots for different types of network traffic.

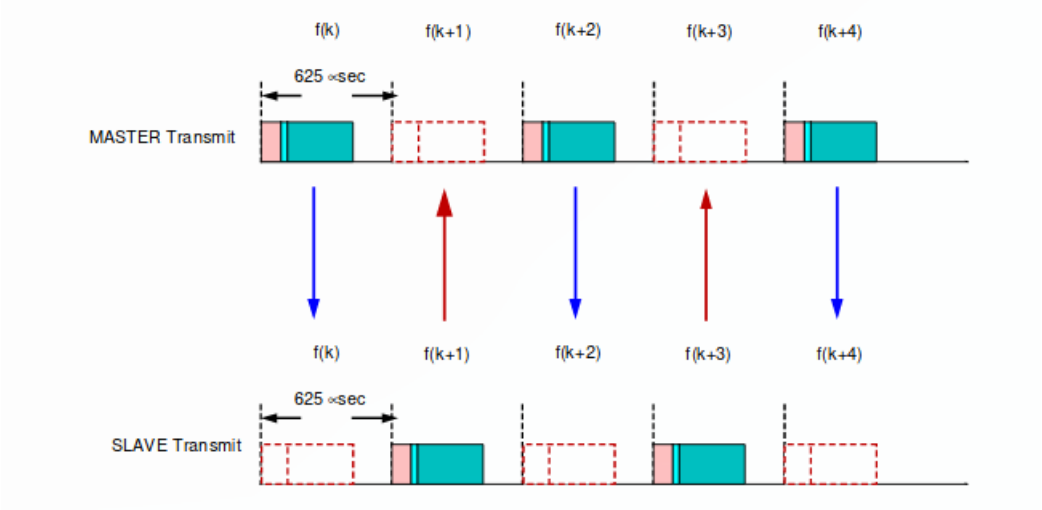

Figure 4.28: TDD illustration [2]

4.8.1.6 Packet Types

BR/EDR packet structure serves as a standard for every packet being transported between two BR/EDR PHY. Packets are designed to allow for optimal use in normal operation cases. Its generic structure can be analyzed in the following scheme.

As various applications and profiles differ, so do the type of traffic that they require to operate. There are several types of packets defined in order to support either data, voice or to support other of Bluetooth's operations.

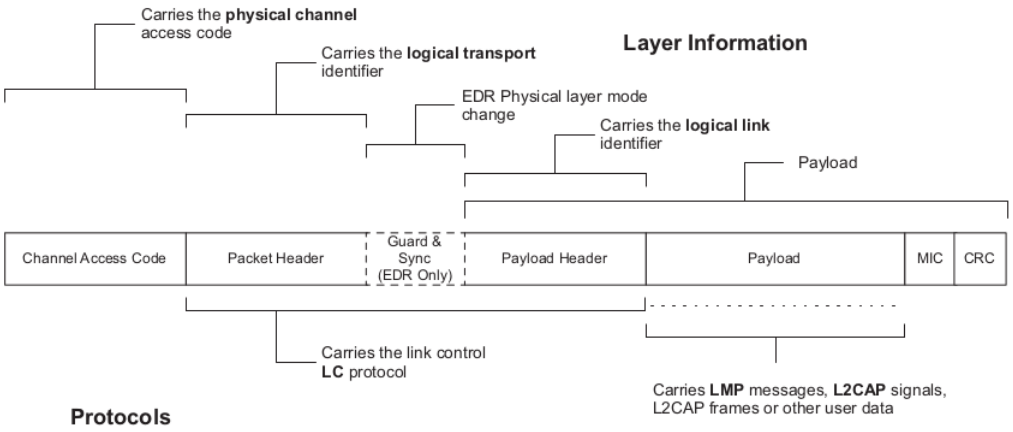

Figure 4.29: BR/EDR packet structure [2]

- ID packet - ID packets are used to convey device access code (DAC) or inquiry access code (IAC), used during inquiry or scan phases.
- NULL packet - A NULL packet is normally used when link layer information is to be conveyed and no other data is buffered to be sent.
- POLL packet - A POLL packet is used by the master in a connection in order to request an answer from a slave.
- FHS packet - Frequency Hopping Sequence packet carries real-time clock information and the Bluetooth device address.
- DM - ACL link packet with FEC encoding. DM packets can be of 1,3 or 5 slots of occupancy and also modulated either at 1, 2 or 3 mb/s, resulting in 9 different DM packets.
- DH - Similar to DM packet, but without FEC. Also features 9 different types of packets.
- AUX1 - AUX1 packet is similar to DH but without any MIC or CRC code.
- DV - Combined data and voice packet used in SCO links.
- HV - Non FEC encoded voice packet similar to DV.
- EV - Extended Voice packet are used in eSCO links, for use in EDR.

Each physical packet contains a header (Fig 4.30) conveying information necessary for the maintenance of connections.

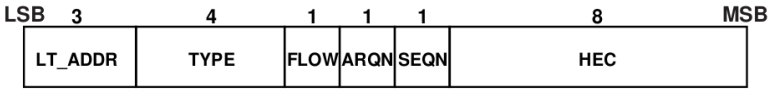

Figure 4.30: BR/EDR Packet Header [2]

The first field, LT_ADDR, is used in order to address individual slaves or broadcast information. The TYPE field identifies the number of slots and modulation scheme used. In order to provide an acknowledgment and request scheme, the ARQN field provides feedback, indicating if a successful transfer was performed or not. Order in the packet stream is supported by the SEQN bit, which provides a alternating numbering scheme enabling the receiving device to identify different packets.

4.8.2 Bluetooth Basic Rate / Enhanced Data Rate PHY

Bluetooth Basic Rate, also known as Bluetooth Classic, was introduced in the first Bluetooth specification, v1.0. The radio specification uses the 2.4 GHz ISM (Industrial, Scientific and Medical) band. Version 1.0 of the Bluetooth specification was quickly deprecated, as problems in that version were resolved. Instead of v1.0, today the oldest active specification is the legacy Bluetooth v2.1. The radio operates with a Frequency Hopping scheme, in order to reduce interference with other devices using the same frequency bands. The frequency hopping scheme was introduce very early in the technology's development, as it is imperative to enable Bluetooth to work in polluted RF environments.

Basic Rate and Enhanced Data Rate differ on the physical data modulation technique. While Enhanced Data Rate (EDR) uses Phase Shift Keying (PSK), Basic Rate makes use of Frequency Shift Keying (FSK).

Basic Rate uses GFSK, a modulation technique described in the following figure. This technique allows for the transmission of 1Msym/s but this throughput is relative to all of the on-air data, not application data.

Enhanced Data Rate makes use of $\Pi/4$-*DQPSK* and 8DQPSK, allowing for double and triple data rate. In a physical channel connecting devices in a network. communication occurs on a TDMA (Time Division Multiple Access) scheme, as the physical channel is divided into time slots.

BR/EDR radios are categorized according to output power within the following 3 classes.

Power Class	Maximum Output Power
1	100 mW
2	2.5 mW
3	1 mW

Table 4.1: BR/EDR Power Classification

4.9 Bluetooth Low Energy Controller Overview

Bluetooth Low Energy is introduced in specifications after 2010 to provide a solution for an emerging set of problems, while trying to keep most of the features that made Bluetooth a proficient wireless communication specification. While BR/EDR addresses problems such as data

rate, LE first approach was enable wireless low power devices, empowering a new generation of devices, and making it easier to develop and implement sensor devices in environments where cables are difficult to use and battery life of those devices is a priority.

4.9.1 LE Link Layer Overview

The LE Link Layer (LL) can be described as a protocol which implements several very important functions over an LE physical channel.

Devices with an LE controller have a 48 bit address that can be either a public or random device address. Public device addresses are generated accordingly to the BR/EDR specification, already mentioned before. Random addresses can be either defined as static or private. Static Random addresses are used if a device is meant to have a different address every time it powers on. Private addresses give the LE Link Layer the possibility to generate encrypted addressing using a public-private key scheme.

In Bluetooth Low Energy, there are three Logical Transports defined, ACL and Advertising Broadcast (ADVB) and Periodic Advertising Broadcast (PADVB). These are used for active connections (ACL), and advertisements (PADVB and ADVB) to be received devices that are listening.

The LL specifies five different states that a LE connection can have:

- Standby State - In this state the Link Layer is idle and does not send or receive packets.
- Advertising State - In the advertising state, only advertisement packets are sent or received.
- Scanning State - This state is only used to listen for advertising channel packets.
- Initiating State - This is a state used with the purpose to listen and start a connection with a specific device.
- Connection State - When in connected state, a device assumes one of two roles: Master or Slave.

Only with these different states Bluetooth LE is able to have such different characteristics than BR/EDR. Only one of the following five LL states is permitted to be active at once. Not all states have to be implemented, allowing manufacturers to implement reduced versions of the Link Layer. The state machine defined is illustrated in the figure below.

4.9.1.1 Connection Interval and Slave Latency

The connection interval is defined in the Bluetooth specification as a period in which two connected devices are required to communicate. It's value starts in 7.5ms and can be as big as 4 seconds, in increments of 1.25ms. While in a piconet, a master is required to send at least one empty packet at each connection interval. A slave response is required every at every connection interval also, but only if another parameter, slave latency is set to zero. If different than zero, the

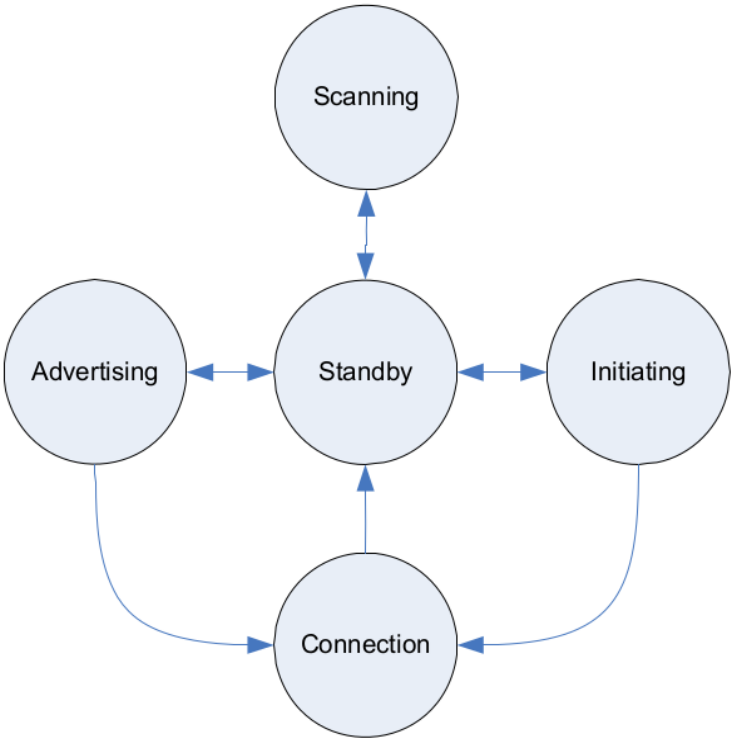

Figure 4.31: LE Link Layer State Machine [2]

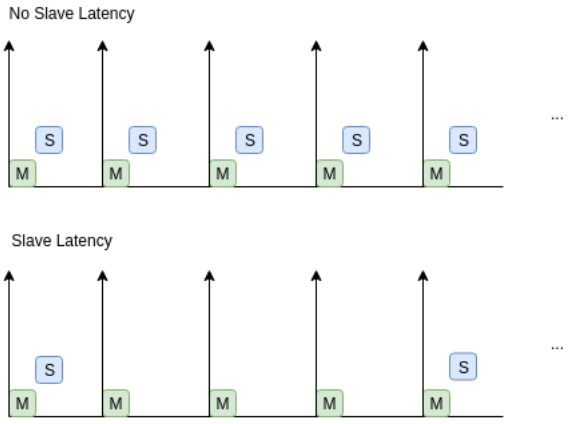

Figure 4.32: Slave Latency Illustration

slave latency parameter is a counter of how many connection intervals a slave can skip, giving the slave a chance to enter sleep mode and save power, an example can be seen in Fig. 4.32

According to Texas Instruments CC2540 Bluetooth low energy Software Developer's Reference Guide [27], changes in the connection interval have the following impact:

- Reducing the connection interval will increase throughput between both devices, reduce latency, and increase power consumption.

- Increasing the connection interval will have the reverse effect, lowering throughput between both devices, increasing latency, and lowering power consumption.
- Increasing Slave Latency will reduce power consumption on the peripheral device and increase it's latency.

4.9.1.2 Connection Event End and Supervision Timeout

In order to inform slaves whether to sleep, in case of a finished connection event, or to keep waiting for another packet from the master, the MD bit of the Data Channel PDU is used. If this bit is not set by either device, after receiving a master packet and sending a response, the connection event is closed, and the slave can sleep up until the next connection event. Even if the slave sets this bit, the master's response is still optional, but the slave must continue listening for a master packet. In case of the master setting this bit, the slave is always required to wait for another packet.

		Master	
		MD = 0	**MD = 1**
Slave	**MD = 0**	Master shall not send another packet, closing the connection event. Slave does not need to listen after sending its packet.	Master may continue the connection event. Slave should listen after sending its packet.
	MD = 1	Master may continue the connection event. Slave should listen after sending its packet.	Master may continue the connection event. Slave should listen after sending its packet.

Figure 4.33: More Data bit usage [2]

In the case of two invalid packet receptions, the connection event is automatically closed.

The Supervision Timeout parameter sets the limit for the maximum time between packets until a the connection is considered dropped. This impacts on how much time a device will try to send data until GAP changes state.

4.9.1.3 Connection setup and operation

The primary procedure for connecting two BLE devices is sending a CONN IND packet to the slave in order to start a connection. Upon receive of the connection request, both devices will start a procedure in order to successfully establish a connection. In this procedure, illustrated in Fig 4.34, there is a waiting window and then a first packet from the master to the slave. If received correctly, the instant that the packet was received marks the start of the connection interval, also known as an anchor point, and both the slave and the master will use this reference to tune into the correct frequency at the correct time, creating a communication link between devices. The time window between the connection request packet and the first connection interval is known

to both devices as it is transmitted in the CONN IND packet and allows the master to schedule communications to this slave while having other radio activities to attend to (such as other slaves, scans and other radio specifications). For each connection interval a different radio channel will be used, but every packet of a determined connection interval will be sent using the same frequency. A master is required to send a packet every connection interval, although it does not have to contain a PDU if there is no data to be sent. A slave, as previously stated does not have to reply to each packet from the master, as a response depends on the slave latency parameter.

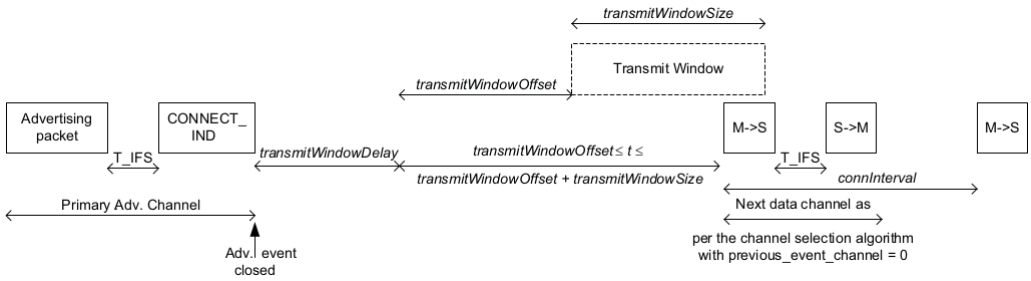

Figure 4.34: Connection Setup [2]

Although there is no specified limit for how many packets can be sent during a connection interval, the number obviously depends on the transmission time of the packets and the connection interval but also on implementation details on both controllers. However, the master has the ultimate control and can refuse to receive more packets from a slave in order to schedule other activities, i.e. if using 0 slave latency, it is only required to reserve the minimum time to send and receive a packet from each slave at each connection event.

4.9.2 LE Baseband

The baseband is composed of 40 radio frequency channels, spreading throughout the frequency spectrum. In order to avoid collisions between devices operating on different networks, an access address present in every packet header. A physical channel is considered connected when two or more devices are correctly tuned to one of the forty defined RF frequencies at the same moment, or more precisely synchronized with timing, frequency and access address. Channels are divided into three categories:

- LE Piconet Channels - Data transmission between devices is handled through this channel.
- LE Advertising Channels - With this physical channel the LE core offers a possibility to broadcast data to unconnected devices, being able to be used to set up connections as well.
- LE Periodic Physical Channels - The channels are used to enable a broadcast between unconnected devices, being responsible for setting the timing in the broadcast communication.

4.9.2.1 LE Packets

There are two types of LE packets, uncoded and coded packets, the later only supported only by the LE 1M PHY. An uncoded packet is composed of several fields, the Preamble, Access Address, PDU and CRC. Each of the 4 fields performs a specific function on the physical layer. The preamble is used for synchronizing devices at the start of the packet, and its size is 1 octet for the LE 1M PHY and 2 for the LE 2M PHY. As referred before, the Access Address is used to avoid collisions if two devices from different networks happen to transmit on the same channel. The PDU carries data relevant to the LL packets. The first specification defined the maximum size of the LE PDU as 23 octets, but since Bluetooth v4.2, the maximum size is 257 octets . CRC is used to check data integrity.

Figure 4.35: LE Uncoded Packet [2]

"The LE Coded PHY allows range to be quadrupled(...)."

Coded packets have a very specific purpose, introduced in Specification 5, these allow for a greater transmission range by applying a Forward Error Correction (FEC) mechanism. Using this method requires the generation of some overhead in order for the receiver to have the ability of correcting a packet in case of errors that occur when receiving a packet, due to either background noise or direct radiation. Two coding schemes are available, creating either 1 or 8 additional bits for each bit input at the FEC generator. As stated, there are two different coding schemes used, determined by the parameter S, either being equal to 2 or 8. These types of packets have a Preamble, Access Address, PDU and CRC fields, but also contain 3 other fields in order to support the described features. Although significant overhead will be produced by this technique, range can be increased without a significant transmission power increment.

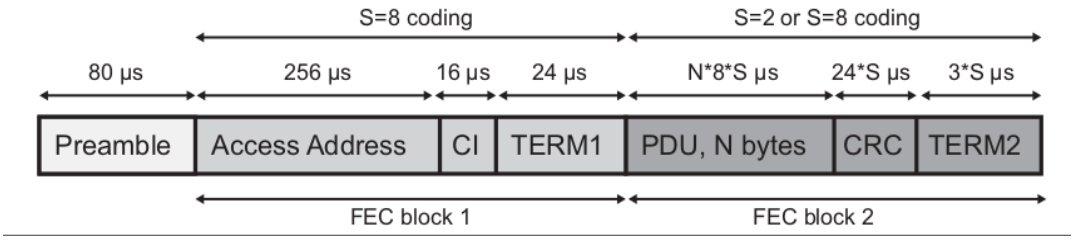

Figure 4.36: LE Coded Packet [2]

4.9.2.2 Data Length Extension

An important feature was introduced in specification 4.2 which enables a packet with a PDU with up to 257 bytes, instead of the default maximum value of 23 bytes. This feature greatly enhances aspects regarding application throughput with BLE devices as it implies less overhead per application data unit.

4.9.3 LE Radio

A Bluetooth LE radio also operates in the same frequency range as the BR/EDR radio, but the differences start with the modulation schemes. When BR/EDR have three different modulation schemes, LE radios have only two with a bandwidth of 1 and 2 Msym/s, using Guassian Frequency Shift Keying (GFSK) as the modulation technique. Time Division Duplex is also used in this radio specification, allowing for duplex communication, where each entity in the connection has a predetermined time slot to communicate. The 1 Msym/s PHY supports two packet formats, coded and uncoded. The output power in LE radios is classified according to the following power classes in Table 4.2:

Power Class	Maximum Output Power
1	100 mW
1.5	10 mW
2	2.5 mW
3	1 mW

Table 4.2: LE Power Classification

4.10 Security

In wireless communications security is always a important topic, as information flows over the air, it can always be captured, jammed, or even injected by other devices. Associated with security is the combination of device parameters in terms of discoverability and connectivity, which defines 3 types of addressing for a Bluetooth device, known as security levels. Also, apart from security levels there are several security modes which defining security procedures to use in each mode. Security in Bluetooth devices is handled by the Security Manager(SM), responsible for configuration and control of several parts of the stack in order to create and maintain secure connections 4.37. As in most parts of the specification, BR/EDR differs a little from LE devices regarding security mechanisms. In order to understand the reasons behind the implementation of security features, one should have in mind several concepts:

- Authentication - Determination of the identity of the remote device.
- Authorization - Determination of whether to allow services to be accessed by remote device.
- Encryption - Obfuscation of data in order to only allow access by authorized parties.
- Privacy - Obfuscation of the device's identity.

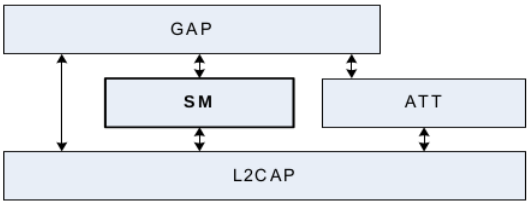

Figure 4.37: Security Manager [2]

The Bluetooth pairing process is divided into several different phases, these provide a structured way to set up a secure connection. These 3 phases are:

- Pairing feature exchange - Determination of the I/O characteristics and security protocol of each device.
- Key Generation - In this phase keys are generated with one of two methods, either LE Secure Connections or LE legacy pairing, depending on device characteristics and authentication requirements.
- Key Distribution - In phase 3, the generated keys are sent to the remote device via Bluetooth or Out of Band (OOB). The choice made in this phase is directly related to phase 1, as device IO capabilities are known in this phase. In Fig 4.39 the different combinations of IO capabilities are shown.

After pairing, the process of bonding two Bluetooth devices enables them to restart connections already encrypted, as the Link Keys are stored in both devices. This reduces security vulnerabilities and makes a possible attack more difficult, if the bonding process was not eavesdropped on.

4.10.1 BR/EDR

For Bluetooth devices with this PHY, there are 4 defined Security Modes, describing behaviour of a device upon connection initiation until it's closure. These different modes are meant to provide security for different use cases, as security sometimes interferes with usability. The different modes are discussed below:

4.10.1.1 Security Modes

There are 4 defined security modes [6], the behavior of a device should be dictated by the chosen mode. Below is a succinct explanation of each one:

- Security Mode 1 - In this mode all devices are considered as non-secure, as it does not require any security features to be used.
- Security Mode 2 - Devices using configured in this mode use a security manager which provides an authorization mechanism at the service level, enabling or not a remote device access to services available.
- Security Mode 3 - This security mode enforces link layer protection, only allowing for physical link establishment if the specified procedures are successful.
- Security Mode 4 - Similar to Security Mode 2, but Secure Simple Pairing (SSP) is used, providing a more robust pairing mechanism.

4.10.1.2 Service Security Levels

In the Bluetooth specification there are also security levels defined, providing requirements for each security mode. For Security Modes 1 and 3 there are no defined requirements, as these modes are either link level security or ,in the case of security mode 1, no security.

For Security Mode 2 authentication, encryption or authorization may be enforced.

In Security Mode 4 the possible enforced requirements are:

- Service Level 0 - No Man in the middle (MITM) protection, encryption or user interaction required.
- Service Level 1 - User interaction required.
- Service Level 2 - Only encryption is required.
- Service Level 3 - Man in the middle (MITM) protection and encryption required.

4.10.1.3 Link Manager Pairing (LMP)

In the early versions of Bluetooth this was the authentication mechanism used, requiring only a 4 digit PIN and a randomly generated secret in order to pair 2 devices. Being a fairly simple approach in terms of cryptography, this mechanism can be easily circumvented with any modern day computer, as there are only 10,000 possible keys. A sequence diagram of the pairing process using LMP is shown in Fig. 4.38.

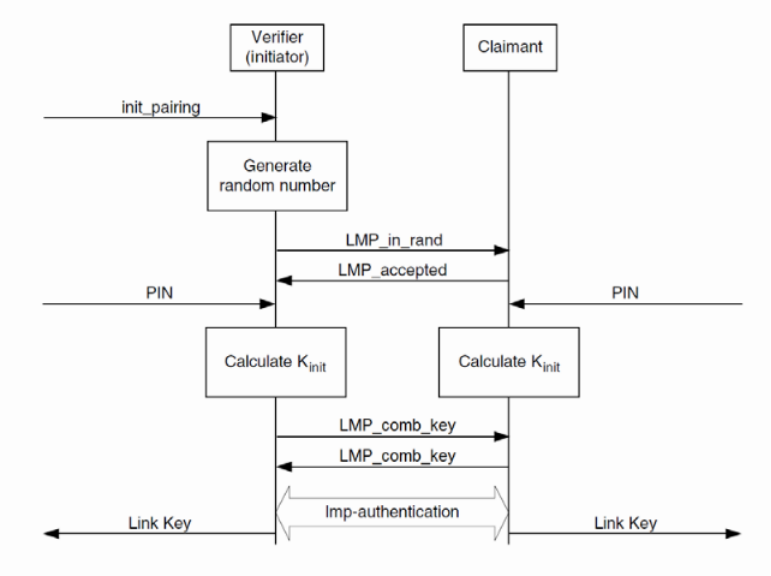

Figure 4.38: LMP Pairing Mechanism [2]

4.10.1.4 Secure simple pairing (SSP)

Introduced in version 2.1+EDR, in order to cover some of the LMP Pairing mechanism flaws, and simplifies the pairing process by providing several association models, covering devices with different IO capabilities, where for LMP a keypad would be required. SSP uses a Diffie-Hellman key as the shared secret between two devices. This mechanism uses a public-private scheme in order to guarantee that no critical information is sent over the air, making it difficult to circumvent even if the pairing process is eavesdropped by an attacker. At the same time, the number of possible solutions is greater than 2^{128}, making it very difficult to compute.

A defining characteristic of this algorithm is that, when a device receives a public key from another, the Link Key is derived from the combination of it's private key with the remote device's public key, and this Link Key is also equal on the remote device's side.

> "Given two SSP key pairs A and B, there exists a well-known function F such that F(PublicA, PrivateB) = F(PublicB, PrivateA). The result of this function is the DHKey. Only the very two devices owning A and B are able to calculate the same DHKey." 4.40

	Initiator				
Responder	**DisplayOnly**	**Display YesNo**	**Keyboard Only**	**NoInput NoOutput**	**Keyboard Display**
Display Only	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated
Display YesNo	Just Works Unauthenticated	Just Works (For LE Legacy Pairing) Unauthenticated Numeric Comparison (For LE Secure Connections) Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): responder displays, initiator inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated
Keyboard Only	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator and responder inputs Authenticated	Just Works Unauthenticated	Passkey Entry: initiator displays, responder inputs Authenticated
NoInput NoOutput	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated
Keyboard Display	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated

Figure 4.39: Pairing regarding IO capabilities [2]

There are 4 association models in SSP, a selection table for possible association models is show in Fig. 4.39:

- Numeric Comparison - A six digit numeric value is show in both devices, and the user is prompted to confirm the number with a yes or no response.

- Passkey Entry - This is used normally if one of the devices has input capabilities, such as a keyboard, and the other output capabilities such as a screen.

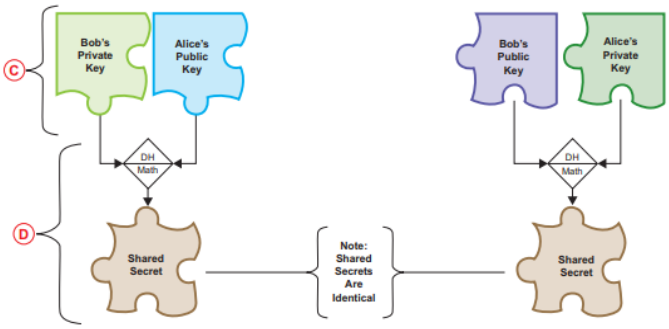

Figure 4.40: Diffie-Hellman Key Generation [5]

- Just Works - In this association model is used when one of the devices does not have IO capabilities. The key received is not checked, as the device automatically accepts it.
- Out of Band - Used normally with NFC or other similar technologies, providing a mechanism to exchange keys with a different technology.

4.10.2 LE

Bluetooth Low Energy implementations are often done in resource constrained devices, as such, security is mostly implemented in a way that does not require much processing power. Mechanisms are somewhat similar to Bluetooth BR/EDR/HS, but several other keys are defined. The notion of Long-Term Key is introduced replacing the Link Key.

Private and Public Addresses were also introduced with LE, providing an effective method for identity tracking prevention, and the Identity Resolving Key, which enables devices to resolve another device's private address.

4.10.2.1 Security Levels and Modes

In Bluetooth Low Energy the security modes provide service-level security, having defined 2 different modes.

- LE Security Mode 1
  - Level 1 - No security, authentication or encryption required.
  - Level 2 - Requires pairing with encryption.
  - Level 3 - Requires authenticated pairing with encrypted
- LE Security Mode 2
  - Level 1 - Requires data signing.
  - Level 2 - Requires authenticated pairing with data signing.

4.10.2.2 Pairing

Pairing is fairly similar, but the procedure is different. Firstly a Temporary Key (TK)is agreed upon and a Short Term Key (STK) is derived. Using the STK, a temporary encrypted link is created in order to securely distribute the other keys: Long-Term Key (LTK), IRK and Connection Signature Resolving Key (CSKR). The analogy can be made that the same role is given to the Link Key in BR/EDR as the Long Term Key in Bluetooth LE. Other keys such as the Identity Resolving Key enable address resolution for devices in which the address is private and resolvable. The Connection Signature Resolving Key enables connections between devices to be authenticated, as the packets sent are signed resorting to cryptographic algorithms also.

The pairing modes have similar names as the ones from SSP used in BR/EDR/HS, although implementation differs. Introduced in Bluetooth v4.0, now known as LE Legacy Pairing, is very similar to SSP but does not use the Diffie-Hellman algorithm in order to generate any of it's encryption keys. This pairing method uses the previously introduced security keys and it's algorithm can be seen in Fig 4.41.

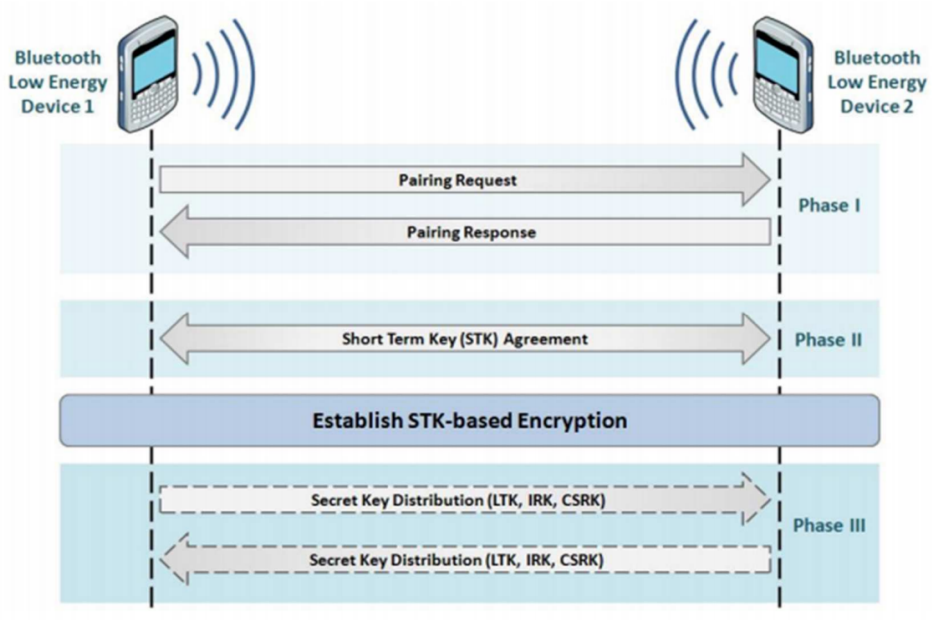

Figure 4.41: LE Legacy Pairing[6]

As it can be seen, the initial encryption used is based on the STK, which is exchanged using one of the four association models similar to those defined in SSP, and as the STK is always exchanged in clear text, a conclusion can be made that this does not provide any eavesdropping protection. An attacker could possibly gather the STK and proceed to sniffing and decrypting other encrpytion keys and further communications between both devices. Introduced in Bluetooth Specification v4.2, LE Secure Connections introduces another pairing method, using the Diffie-Hellman algorithm, provides a more secure approach to Bluetooth Low Energy pairing.

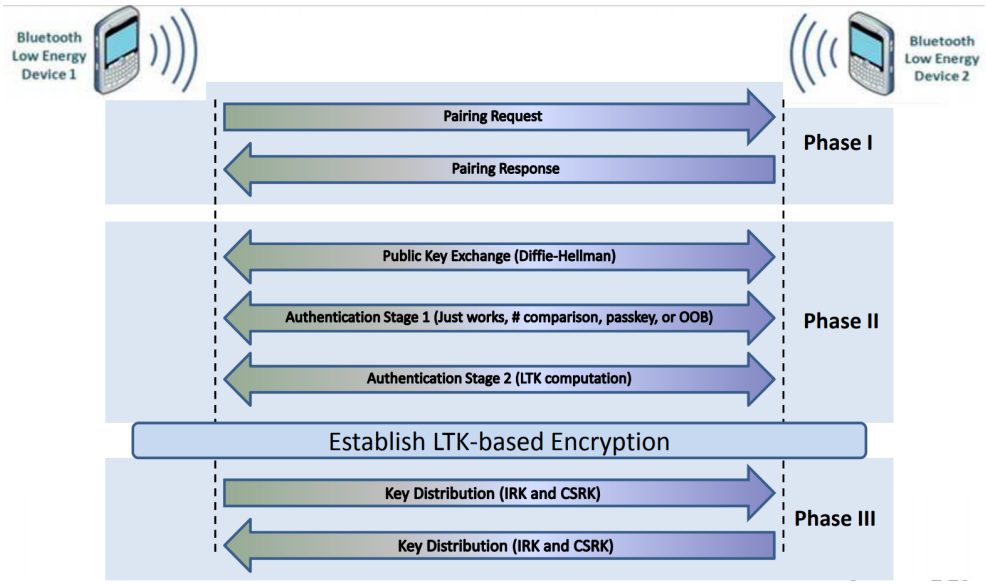

Figure 4.42: LE Secure Connections Pairing[6]

Using method and benefiting from asymmetric key generation [5], keys are generated securely and no critical data is shared prior to encryption, and the LTK, IRK an CSRK are only shared after, within an encrypted connection.

4.11 Throughput Considerations

In this section theoretical calculations of maximum throughput will be presented, as well a discussion about overal Bluetooth performance. First of all, the reader has to have in mind the concept of goodput, which is understood as the application throughput. Although many Bluetooth modules are advertised as having data rates as high as the baseband bit rate, this does not translate into application throughput, as there are several sources of communication overhead, naturally.

As a review about Bluetooth Technology was made in the previous sections, it can be observed that the different specifications support physical layers with different data rates, i.e. Bluetooth Basic Rate and Bluetooth Enhanced Data Rate have two different data rates, 1 mb/s and 2/3 mb/s, respectively. As data passes through several stack layers and respective buffers, delivery to the application is not immediate and dependent on stack implementation and RF conditions.

4.11.1 Bluetooth Classic

Achieving maximum goodput will always depend on the amount of stack layers underneath an application. For example, audio data transmitted over SCO links is delivered directly to the HCI layer, allowing for lower latency. On the other hand other profiles using RFCOMM as the main carrier are influenced by protocol overhead. The best case scenario for an application that does not use an SCO link is to use directly an L2CAP channel, configuring it's parameters in a way that benefits application throughput.

Theoretical values for throughput, regarding ACL links and different types of packets, were withdrawn from the Bluetooth Specification 5, as can be seen in Fig.4.44:

Type	Payload Header (bytes)	User Payload (bytes)	FEC	MIC	CRC	Symmetric Max. Rate (kb/s)	Asymmetric Max. Rate (kb/s)	
							Forward	Reverse
DM1	1	0-17	2/3	C.1	Yes	108.8	108.8	108.8
DH1	1	0-27	No	C.1	Yes	172.8	172.8	172.8
DM3	2	0-121	2/3	C.1	Yes	258.1	387.2	54.4
DH3	2	0-183	No	C.1	Yes	390.4	585.6	86.4
DM5	2	0-224	2/3	C.1	Yes	286.7	477.8	36.3
DH5	2	0-339	No	C.1	Yes	433.9	723.2	57.6
AUX1	1	0-29	No	No	No	185.6	185.6	185.6
2-DH1	2	0-54	No	C.1	Yes	345.6	345.6	345.6
2-DH3	2	0-367	No	C.1	Yes	782.9	1174.4	172.8
2-DH5	2	0-679	No	C.1	Yes	869.1	1448.5	115.2
3-DH1	2	0-83	No	C.1	Yes	531.2	531.2	531.2
3-DH3	2	0-552	No	C.1	Yes	1177.6	1766.4	235.6
3-DH5	2	0-1021	No	C.1	Yes	1306.9	2178.1	177.1

Figure 4.43: BR/EDR ACL link packet summary

In Fig. 4.44 can be seen encapsulation for an L2CAP connection-oriented and connection-less channels.

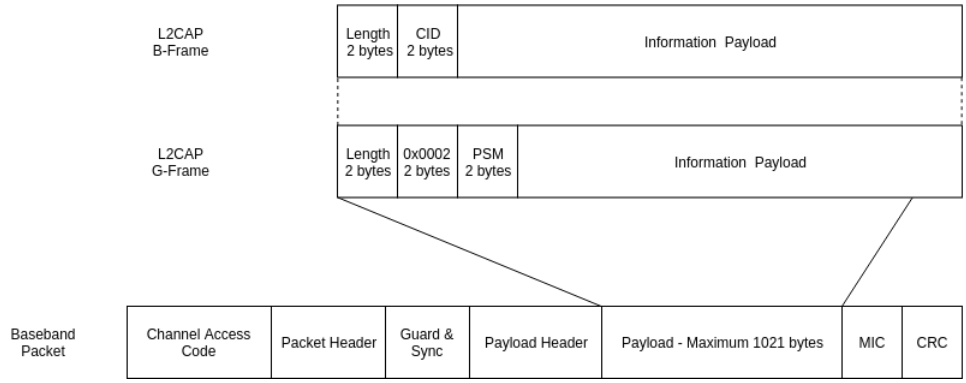

Figure 4.44: ACL link EDR packet summary

There are two types of payload headers for ACL packets, one used in single slot packets and another for multi-slot packets. For the single slot packets, the payload header is composed of 3

fields, the Logical Link ID (LLID) is used to identify the link, and in case of an ACL-U or ASB-U if it is the start or continuation of an L2CAP message. The second field is only a bit long and is name FLOW, intended to control whether if the remote device can send more ACL packets, although specific links have different usages for the FLOW bit. The final field in ACL single-slot packets is the length, referring to the length of the payload. Using a 3-DH5 packet, one would theoretically be able to achieve maximum for an application data transfer rate if using an L2CAP connection oriented channel, 2.1mb/s. Although this is all theoretically achievable, actual payload normally embeds other protocol data such as the RFCOMM protocol, used by many profiles.

As shown is previous sections the RFCOMM Protocol, adapted from TS 07.10, was developed to support virtual serial port. RFCOMM communications are divided into packets, having the format shown in Fig. 4.45.

Address	Control	Length Indicator	Information	FCS
1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet

Figure 4.45: RFCOMM Frame Structure

As dictated in RFCOMM Specification v1.2 [4], the default allowed frame size is 127 bytes and supporting an interval from 23 to 32767.

Experiences made by manufacturers such as Texas Instruments have been made with the purpose of testing their hardware. A test was performed using the Serial Port Profile in order to communicate data between an embedded Bluetooth module to a PC [28]. The maximum achieved throughput was 675kbit/s, using a UART bit rate of 921kbit/s.

Although this data rate was achieved by Texas' specific hardware, not all implementation are required to perform as well, as compliance requirements are much lower and SPP is intended to be a profile to support legacy applications, SPP was not developed in order to achieve a high throughput.

> "This profile requires support for one-slot packets only. This means that this profile ensures that data rates up to 128 kbit/s can be used. Support for higher rates is optional."

4.11.2 Bluetooth Low Energy

In Bluetooth Low Energy, communications are also performed in a master/slave schema and happen within connection intervals. The start of a connection interval is referred to as an *anchor point*, where it is expected for the master to send a starting packet, that may not contain a payload, to the

slave. This marks the start of the connection interval, after this point the master listens to a reply by the slave. Depending on the configured slave latency, the slave may or may not be required to reply with a packet. The connection interval continues until one of the devices signals that it does not have more data to send. In between packets from the master and the slave there is a required window of 150μs, known as the inter-frame space (T_IFS), where no device may transmit.

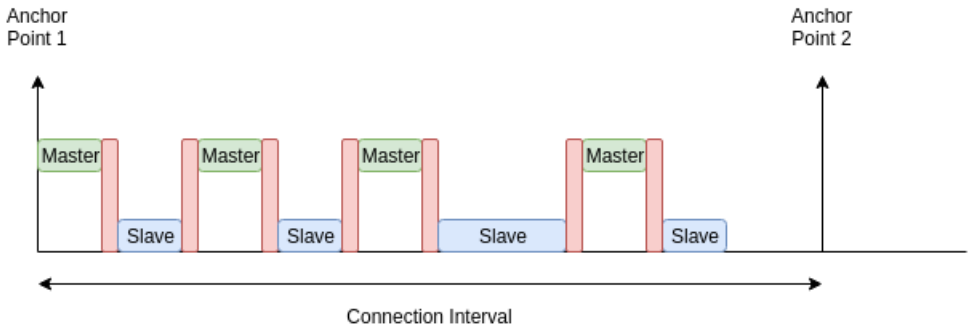

Figure 4.46: Connection Interval Illustration

Summarizing ***Exploring Bluetooth 5 - How Fast Can It Be?*** [29], packets are different for Bluetooth v4.0/v4.1 and v4.2. As seen in Fig. 4.48, the length field in Bluetooth v4.2 is a byte long, while for v4.0/v4.1 is just 5 bits long.

Also, referring to ***Exploring Bluetooth 5 - How Fast Can It Be?***, one can understand the definition of throughput as:

$$Throughput = \frac{payload}{period}$$

In a Bluetooth Low Energy connection even if application data is only being sent from slave to master, the master still has to send packets to the slave in order to control piconet traffic. As such, optimal throughput can be calculated for each of the different specifications:

- In Bluetooth v4.0/v4.1 the maximum permited L2CAP payload size is 27 bytes, being that the total packet size will be:

  $$DataPacketSize = Preamble + AccessAddress + PayloadHeader + Payload + Mic + CRC = 41bytes$$

  $$NULLPacketSize = Preamble + AccessAddress + PayloadHeader + CRC = 10bytes$$

  Transmission the biggest packet results in 328μs of transmission time. As the master polls slaves with NULL packets, and these do not contain any payload or MIC fields, total transmission time will be 80μs.

  With this in mind, the total period for transmission and reception would be:

  $$TotalPeriod = (80 + 150 + 328 + 150) = 708\mu s$$

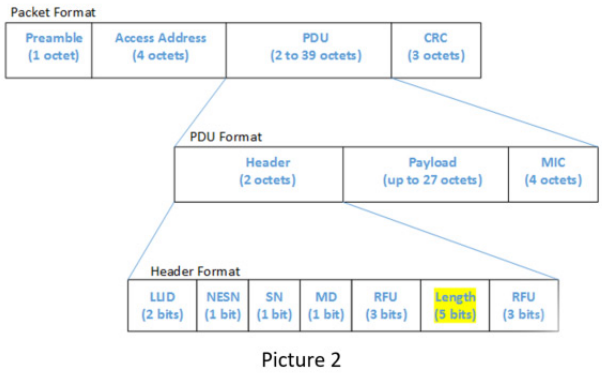

Figure 4.47: Bluetooth v4.0/v4.1 packet format

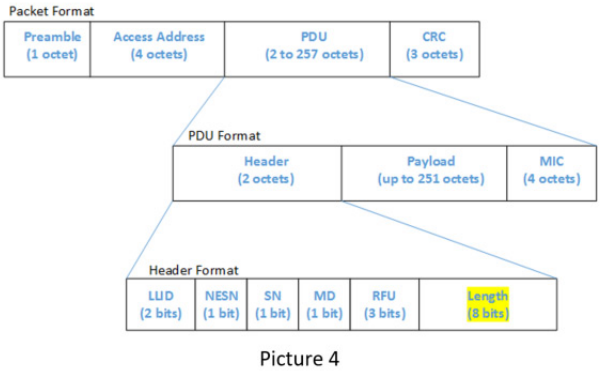

Figure 4.48: Bluetooth v4.2 packet format

Calculating maximum throughput for this connection would result in:

$$Throughput = \frac{27 \times 8}{708} \times 10^6 = 305,084 kbit/s$$

However this is the throughput for the data sent over the Link Layer. As applications normally use GATT/ATT, additional overhead as to be taken into account. As this article calculates[30], throughput for applications using GATT is affected by 4 additional L2CAP bytes and 3 ATT protocol bytes, in the case of a notification. It is also stated that application throughput if using GATT and a 7,5ms connection interval is approximately 236,7 kbit/s, however implementations of different Bluetooth Controllers and Stacks sometimes impose limits to the number of packets being sent during a connection event.

- For Bluetooth v4.2, if using Data Length Extension, the payload field length can be negotiated up to 251 bytes. This results in a maximum transmission time of 2120 μs and therefore the transmission/reception period increases to 2500μs. Maximum Link Layer throughput is then equal to 803 kbit/s.

- Regarding Bluetooth 5 and using the LE 2M physical layer, packets are transmitted at double the rate, although the interframe space remains the same, 150 μs . With this in mind,

maximum throughput can be calculated as:

$$Throughput = \frac{251 \times 8}{(40+150+1060+150)\mu s} \times 10^6 = 1,4mb/s$$

[29]

One important consideration is that throughput varies with the connection interval as. As shown in the previously mentioned article [30], we can observe the throughput in function of the connection interval, suggesting that a lower connection interval allows for higher throughput. At Apple's World Worldwide Developers Conference (WWDC) 2017, during the *What ' s New in Core Bluetooth Best practices [31]* presentation about the Core Bluetooth API, a recommendation was made to use a 15ms connection interval in order to maximize throughput .

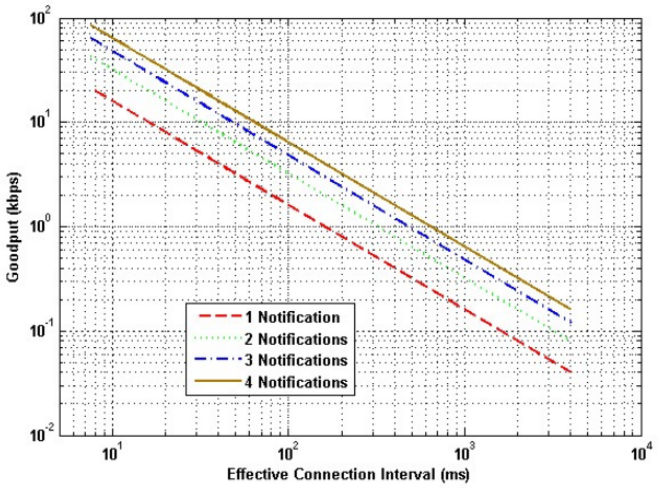

Figure 4.49: Application throughput in function of the connection interval

Summarizing, a graphic of BLE's throughput regarding features and different versions of the Bluetooth specification can be seen in Fig. 4.50.

4.12 Summary

In the past chapter, an more detailed view about the protocols used in Bluetooth Classic and BLE have been discussed, as well as the most important features regarding security, throughput and their characteristics. While Bluetooth Classic supports higher data rates, including more logical transports and supporting two network topologies, Bluetooth Low Energy aims at the requirements of small battery powered devices. Throughput was not a primary concern when first developing BLE but, as the technology evolved more effort has been make in order to enhance bandwidth.

The influence of the connection parameters, such as the connection interval and packet length was discussed. A lower connection interval allows for bigger throughput partly due to the fact that

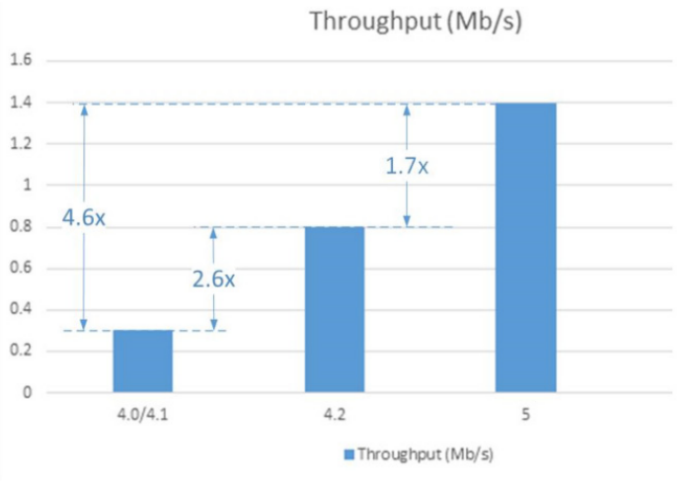

Figure 4.50: BLE Throughput Chart
[29]

most stack implementation limit the number of notifications per connection event. The difference that the Link Layer packet length has regarding throughput was also shown, demonstrating that the Data Length Extension feature introduced in Bluetooth v4.2 allowed for an increase of over two fold of throughput when using the 1M PHY, and over 4 times when using the 2M PHY. The maximum theoretical data rates for Bluetooth Classic, LE 1M and 2M are, respectively, 2,1 mb/s, 803 kbit/s and 1,4 mb/s.

Chapter 5

Communications for Prosthesis fitting

In this chapter a series of practical experiments will be described, first explaining the theoretical concepts that lie behind the implementation followed by a discussion.

Various test were carried out, using a Bluegiga WT12 Bluetooth v2.1+EDR module, a Microchip BM78 (Bluetooth v3.0 with EDR) module, and two other Bluetooth Low Energy *SoCs*, the ESP32 and the CC2640R2. In order to test relative throughput, two different versions of the iPad and a Desktop PC were used. BR/EDR and BLE connections were analyzed, resorting to Wireshark and an Ubertooth, and testing different modules in order to have information about their performance.

Using the Ubertooth platform, only BLE connections were successfully captured, as captured BR/EDR payloads are encrypted and no tool was found in order to decrypt them (Fig. 5.1). Consequently, no traffic between the iPad and any of the Bluetooth Classic modules was successfully captured.

An additional demonstration was performed, using the ESP32 communicating over WiFi to a TCP server, in order to measure this system's capabilities when resorting to WiFi for wireless communication and provide a comparison whilst to tests performed with Bluetooth Classic and Low Energy.

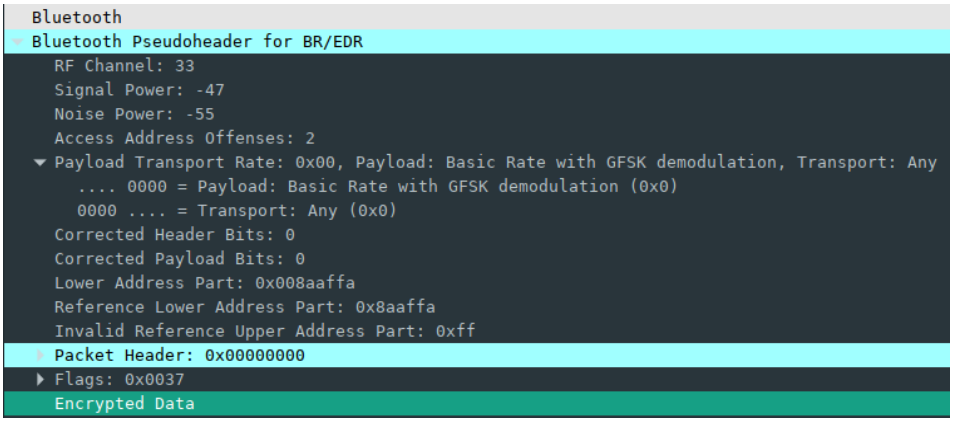

Figure 5.1: BR/EDR captured packet

5.1 Preliminary Experiments

The connection between the iPad and BM78 is only possible to capture if using expensive and professional hardware, however, if connecting it to a GNU/Linux machine an insight into connection parameters and features of the BM78 can be retrieved. BlueZ, the official Bluetooth stack used by the Linux kernel, offers an interface that allows programs, such as Wireshark and Kismet, to access the communications handled by the respective stack. Capturing the traffic between a PC and the BM78 could provide some clarification as to what is the cause for the problems described earlier, and therefore show if the Bluetooth module is indeed the source of the problem or if it is related with iAP and the iOS kernel management.

Preliminary tests were performed with the BM78 and WT12 modules, connected to a PC, in order to be able to analyze connection setup and Link Manager parameters. As the only mainstream OS currently supporting Bluetooth Wireshark captures is GNU/Linux, the OS used in order to perform these tests was Arch Linux (4.16.12-2-ck).

Tests were performed with a Linux desktop with Realtek's RTL8820 WiFi/Bluetooth module, resorting to a *Python* program that was specifically developed in order to communicate with the *tInterface*, enabling it to receive data streamed from the device and evaluate throughput at the application level.

Using Wireshark, it was possible to capture and analyze the connection between the two devices. Packets were captured at the HCI level and consequently all of the communications between the Bluetooth host and controller were accessible. In this way, two independent sources were used to evaluate the connection throughput.

```
________Test Report________

Test Time:  0 : 43
Total Messages Received:  2171
TimeStamps Skipped =  62
Message Errors =  3796
Biggest time gap (ms) =  60
Total time gaps (ms) =  2620
Total Bytes Received:  379093
Average Troughput:  68.99 Kb/s
Sending Stop Message
Connection Closed!
```

Figure 5.2: Python Program Output

5.1.1 Microchip BM78

The aim of this experiment, using the Bluetooth module currently present in the *tInterface*, was to evaluate if the problems previously referred were indeed dependent on the Bluetooth module itself. Connecting it to a Linux PC with an open-source Bluetooth stack, the access to logs, documentation and support is more widely available than for closed source software. As such, the initial connection setup was analyzed and, after that the throughput was evaluated resorting both to the *Python* program and Wireshark. Three different tests were performed, using a sampling period of 20, 15 and 10ms.

5.1.2 Connection Setup

Wireshark packet captures from tests performed have shown how the beginning of a connection between the desktop PC and the BM78 module is done. At first the PC sends an HCI command to the controller in order to create a connection (Fig. 5.3) and, after an event occurs reporting success, the maximum number of slots is extended to 5 (Fig. 5.4). Following this, supported features are exchanged and pairing and encryption procedures are completed. The master of the connection at this point is the PC. After this, a request for the remote's supported LMP is performed and after that L2CAP features. Following this exchange an RFCOMM connection is created (Fig. 5.5) and a Data Link Connection (DLC) is created and configured , resorting to the multiplexer control connection (DLC 0) The remote's SDP attributes are received, after the creation of an L2CAP channel to the SDP service. At this point application data starts being sent over another channel multiplexed over RFCOMM.

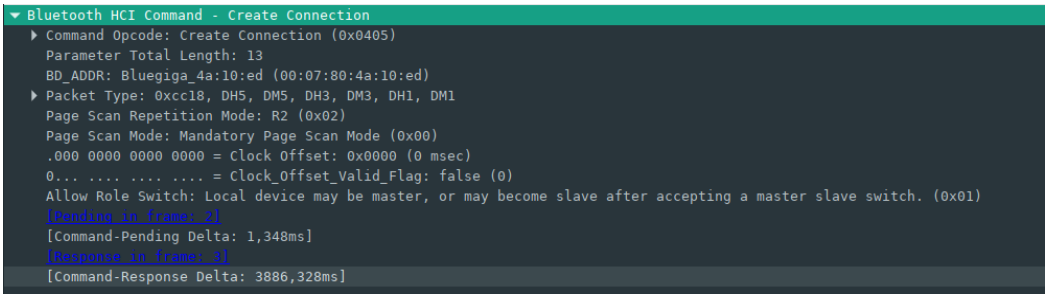

Figure 5.3: Initial Host connection request

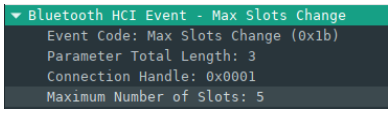

Figure 5.4: Maximum slots per packet update

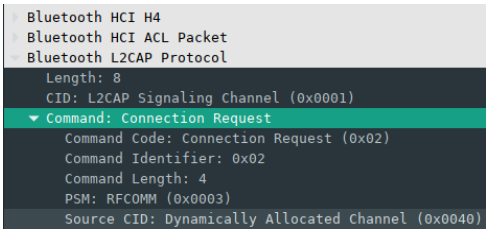

Figure 5.5: RFCOMM connection request

After the maximum slot negotiation and security procedures, the RFCOMM connection where the application data will be transmitted is created. As defined in the RFCOMM specification, the Data Link Connections are defined, with an acknowledgment timer of 0ms, a maximum number of retransmissions of 0 (Fig. 5.6). This means that reliability of the connection is only provided

at lower levels of the stack, as no acknowledgment or retransmission is enabled in the RFCOMM layer.

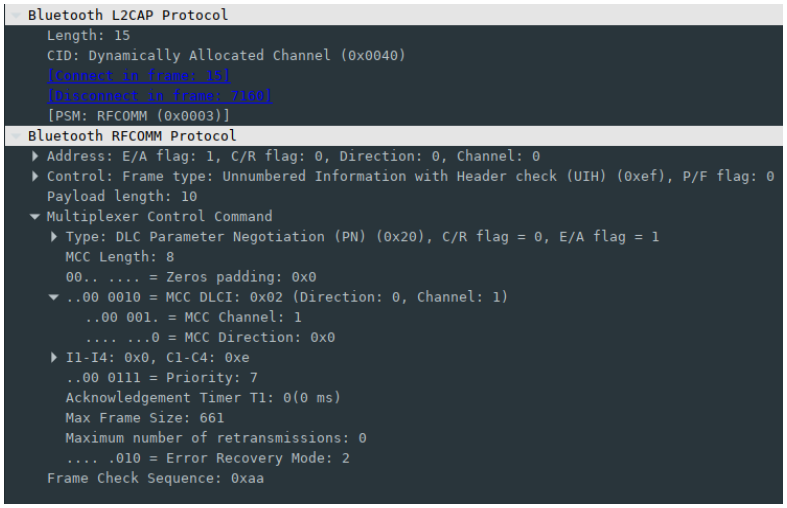

Figure 5.6: PC RFCOMM connection request

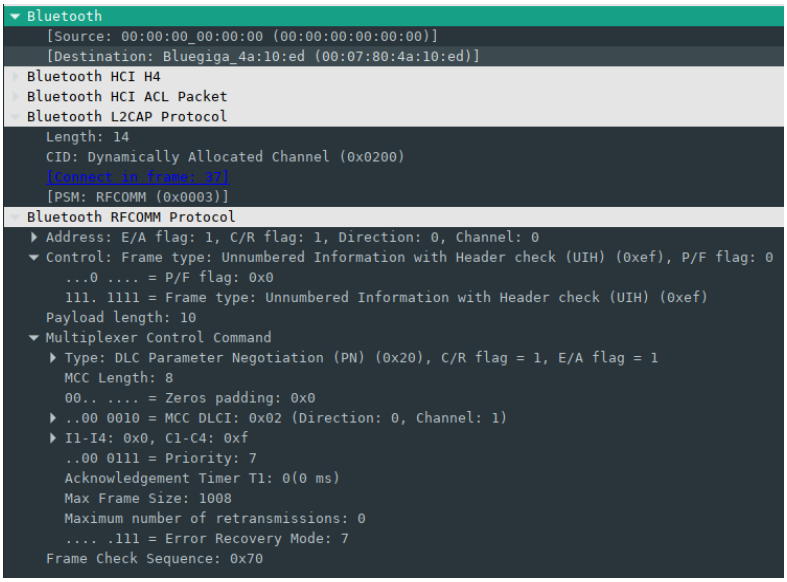

Figure 5.7: RFCOMM Channel 1 parameters

After this, an L2CAP channel is formed in order to read the WT12's SDP records and, afterwards, serial data starts being sent over DLC 1.

It is important to notice, at this point, that the reliability of communications is delegated by the RFCOMM protocol to the lower layers, L2CAP and the Link Manager Protocol.

> "Data frames do not require any response in the RFCOMM protocol, and are thus unacknowledged. Therefore, RFCOMM must require L2CAP to provide channels with

maximum reliability, to ensure that all frames are delivered in order, and without duplicates. Should an L2CAP channel fail to provide this, RFCOMM expects a link loss notification, which should be handled by RFCOMM as described in Section 5.2.3."

Also, as RFCOMM specifications later than version 1.0B require that a credit based control flow is to be used for communications with this protocol. The credit based flow control imposes that for each message sent by a slave device, a credit has to be used and, if there are no more remaining credits, no messages can be sent. Therefore, the master periodically sends a UIH frame with conveying credits, enabling the slave to send data at the RFCOMM level (Fig. 5.8).

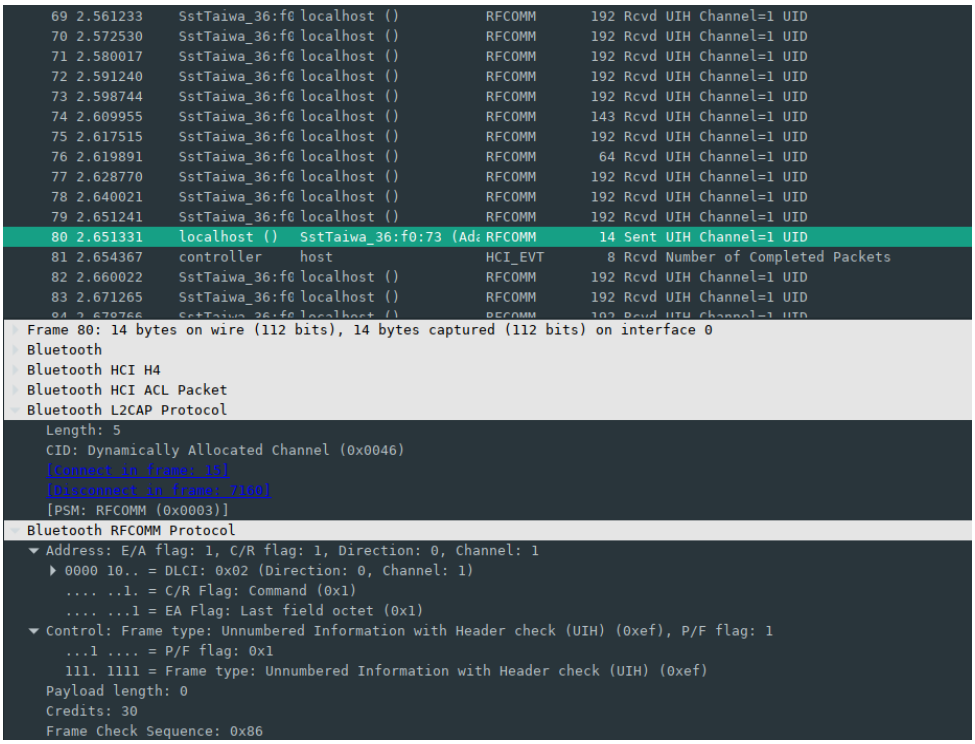

Figure 5.8: RFCOMM flow control

5.1.2.1 Results

Three tests were performed, lasting 1 minute and with different periods for the UART application messages (*tInterface* sampling rate). In the following tables, results from the *Python* program and Wireshark can be seen:

Period	20 ms	15 ms	10 ms
Achieved Tp.	69,22 kb/s	93,4 Kpbs	101,87 kb/s
Required Tp.	70,8 kb/s	94,4 kb/s	141,6 kb/s
Messages Lost	169	369	2244
Timestamp Gaps(Max)	280ms	405ms	460ms
Timestamp Gaps(Total)	3380ms	5535ms	22440ms

Table 5.1: Python Program Test Results (BM78)

Period	20 ms	15 ms	10 ms
Packets	6778	7923	16667
Avg. packets/s	96,373	113,535	259,536
Avg. packet size	90,852	102,499	61,357
Bytes	615792	812096	744029
Avg. bytes/s	8755,631	11637,223	15924
Avg mb/s	0,070	0,093	0,127

Table 5.2: Wireshark Results Summary (BM78)

Throughput graphics generated in Wireshark can also be consulted in Figs. 5.9, 5.10 and 5.11.

- Wireshark I/O Graph, test period 20ms (Bits/second):

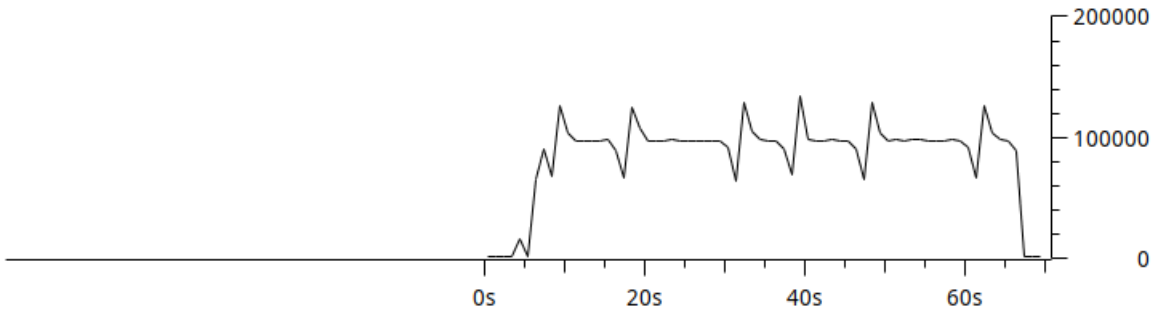

Figure 5.9: BM78 Throughput test - 20 ms

- Wireshark I/O Graph, test period 15ms (Bits/second):

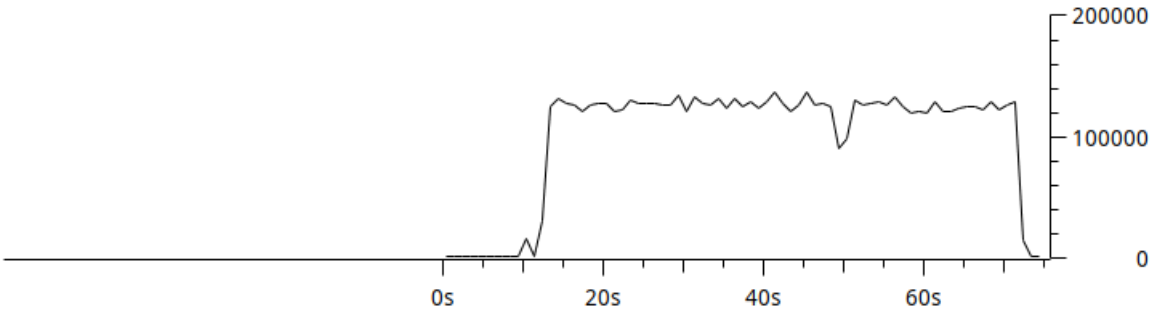

Figure 5.10: BM78 Throughput test - 15 ms

- Wireshark I/O Graph, test period 10ms (Bits/second):

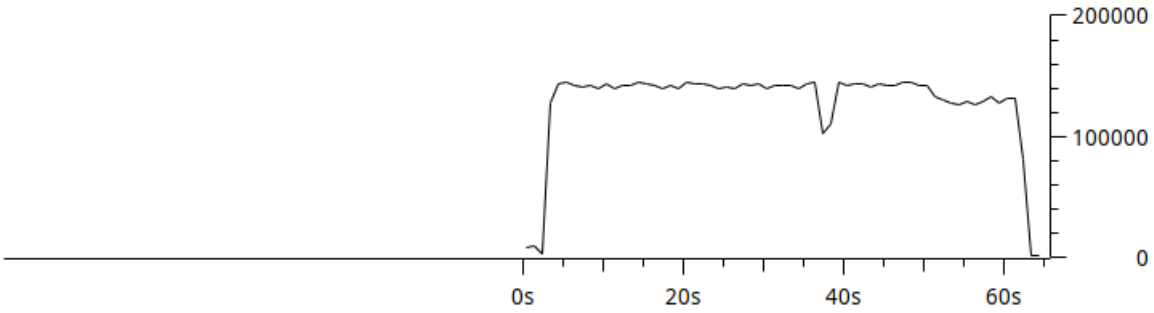

Figure 5.11: BM78 Throughput test - 10 ms

As it is clear from the results, this module delivers an considerable amount of data with errors to the application level and, analyzing the throughput graphics from Wireshark some erratic behaviour can be seen, such as the spikes in the throughput and occasional moments where communication ceases at the RFCOMM level. Tests were repeated several times and results sometimes were significantly worst than the ones shown above. The BM78, whilst operating at these data rates, displays different behaviours when repeating tests several times, given this, it can be observed that this module indeed does not support, in a stable manner, these data rates. In the following images (Figs. 5.12,5.13,5.14), the different behaviour for tests performed with the same 10ms application message period can be seen.

Figure 5.12: BM78 Throughput test, 10ms

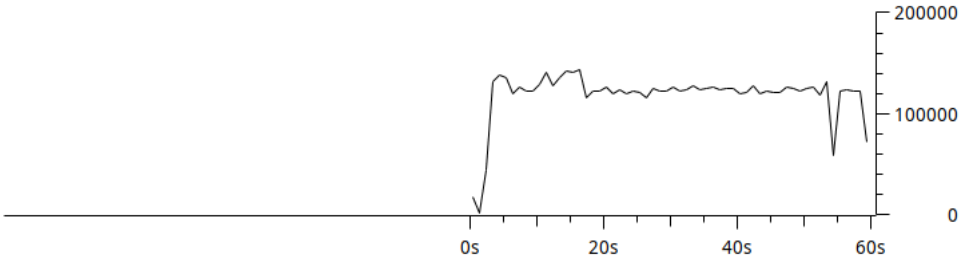

Figure 5.13: BM78 Throughput test, 10ms

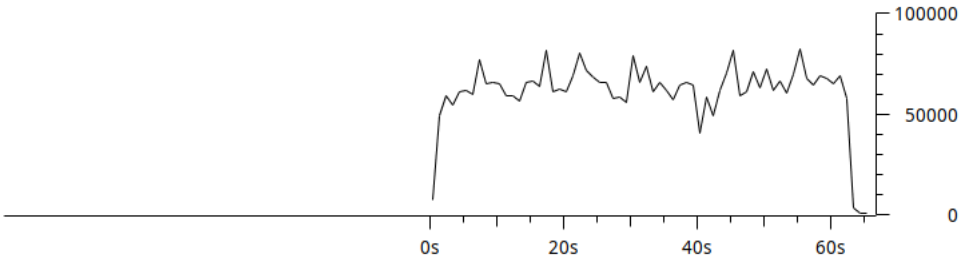

Figure 5.14: BM78 Throughput test, 10ms

The lack of control over the BM78's behaviour was the defining point at which it was decided that a deeper understanding of the Bluetooth specification was required. The many messages containing errors, which are discarded at the application level, are also a problem, as implementing a higher layer acknowledge repeat-request (ARQ) scheme would impact throughput even further. As the flow control between L2CAP and the RFCOMM protocols is not defined in any of the specifications and is implementation-specific, no assumptions can be made as the BM78's implementation details are not known. This module is not only unable to provide the required throughput as it is to implement other solutions that would require more control of the stack. Being very restrictive in terms of usage and configuration, the BM78 is therefore shown as not viable for fulfilling this application's specific needs.

5.1.3 WT12

Without prior knowledge about the limitations imposed by Apple's iAP, the first attempt at solving this problem was finding a Bluetooth v2.1+EDR module that would allow precise control of its stack parameters and configurations. Most Bluetooth modules in the market reside under two categories, either a module that already implements a Bluetooth Core Controller and Host in one package and provides a simple interface for data transport (using SPP or a similar profile) or, a module that only implements a Bluetooth Controller and providing an interface to the Host. As the purpose of this preliminary experiment is to test the possibility of using BR/EDR as a solution, a module which contains both the Bluetooth Core Controller and Host was chosen. Although only the Serial Port Profile would be necessary in order to provide an interface for data transport, the chosen module, Bluegiga's WT12 implements 12 different profiles, such as iAP, HDP and HID.

In order to provide a good interface for the user, the WT12 runs iWrap, a proprietary Bluetooth stack, which is controlled via UART commands and provides a straight-forward way to enable and configure the different available profiles.

Substituting the BM78 by the WT12 Bluetooth module, a small portion of code was produced to properly configure the WT12 module to select the SPP profile and appropriate baud rate. An Atmel SAML21J18 Xplained Pro Board and a WT12 breakout board were used in order to perform testing. The system was mounted using the same configuration as the previous one, a UART bus between the Bluetooth module and the microcontroller.

Six tests were conducted, each with a different period for the task running in the SAM microcontroller, with the duration of one minute. In the next paragraphs, test results produced with Wireshark and the program written in *Python* will be presented. A sample output of the Python program can be seen in Fig. 5.2.

5.1.3.1 Results

In the following table, 5.3, a comparison of the required values of throughput for the different sampling periods and the test results.

Period	20 ms	15 ms	10 ms	5 ms	2,5 ms	Continuously
Achieved Tp.	70,22 kb/s	93,4 kb/s	140,29 kb/s	280,46 kb/s	559,07 kb/s	627,7 kb/s
Required Tp.	70,8 kb/s	94,4 kb/s	141,6 kb/s	283,2 kb/s	566,4 kb/s	921,6 kb/s
Msgs. Lost	0	0	0	1	113	787
Timestamp Gap(Max)	0	0ms	0ms	5ms	22,5ms	50ms

Table 5.3: Python Program Test Results (WT12)

Period	20 ms	15 ms	10 ms	5 ms	2.5 ms	Continuously
Packets	6778	7923	9524	14389	18063	1852
Avg. packets/s	96,373	113,535	136,917	209,312	233,301	110,136
Avg. packet size	90,852	102,499	124,823	159,810	244,084	277,682
Bytes	615792	812096	1188811	2299504	4408890	514267
Avg. bytes/s	8755,631	11637,223	17090,352	33450,058	56945,014	30582,769
Avg mb/s	0,070	0,093	0,137	0,268	0,456	0,245

Table 5.4: Wireshark Results Summary (WT12)

- Wireshark I/O Graph, test period 20ms (Bits/second):

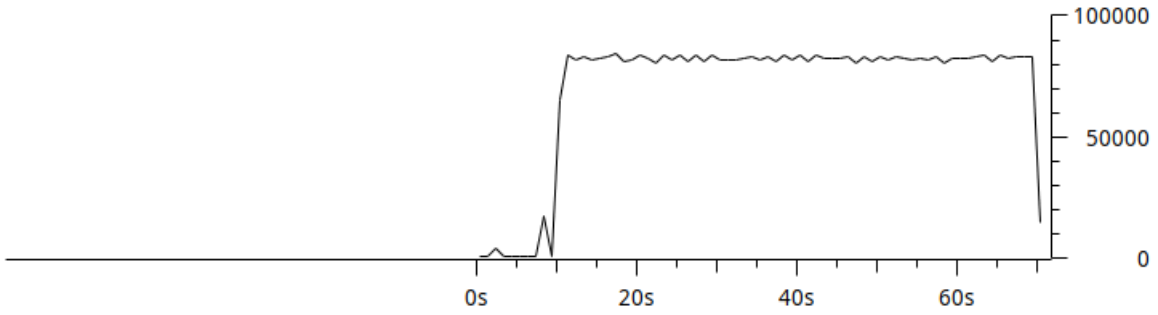

Figure 5.15: WT12 Throughput test - 20 ms

- Wireshark I/O Graph, test period 15ms (Bits/second):

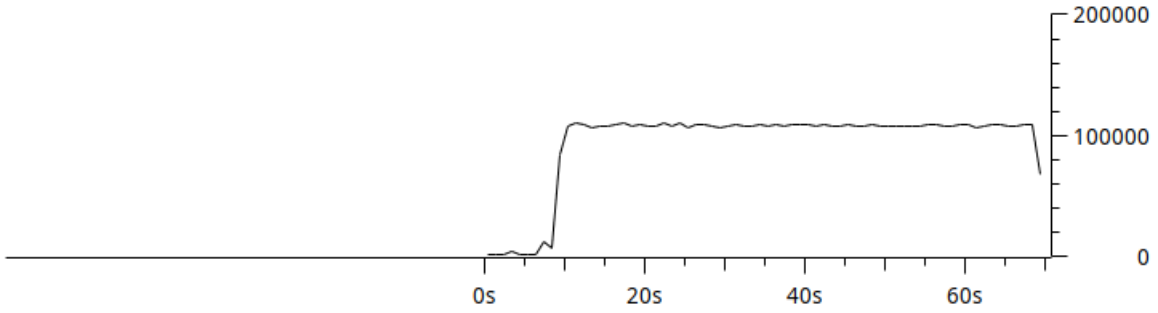

Figure 5.16: WT12 Throughput test - 15 ms

- Wireshark I/O Graph, test period 10ms (Bits/second):

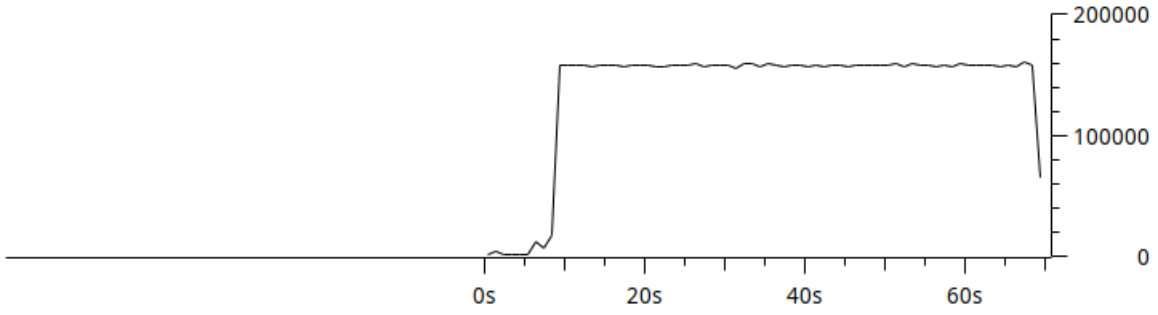

Figure 5.17: WT12 Throughput test - 10 ms

- Wireshark I/O Graph, test period 5ms (Bits/second):

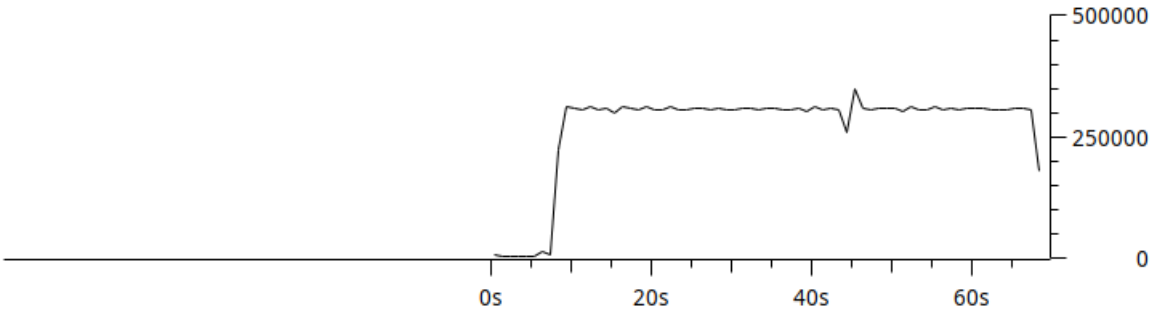

Figure 5.18: WT12 Throughput test - 5 ms

- Wireshark I/O Graph, test period 2,5ms (Bits/second):

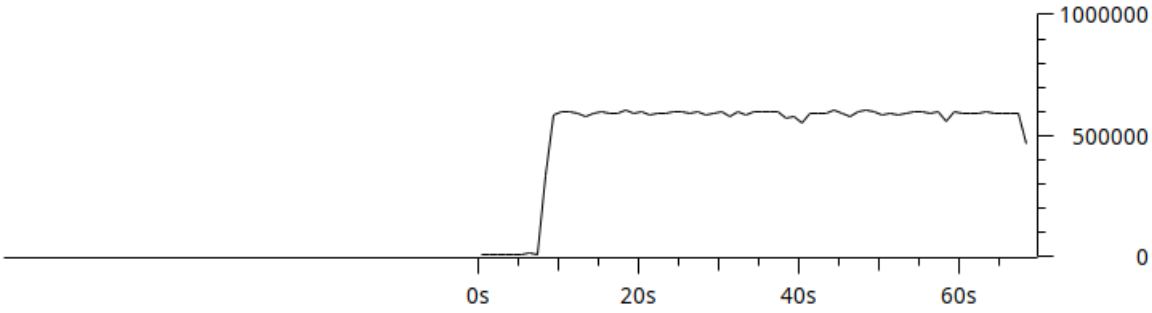

Figure 5.19: WT12 Throughput test - 2,5 ms

- Wireshark I/O Graph, continuous test (Bits/second):

Figure 5.20: WT12 Throughput test - Continuous

It is important to refer that when performing tests sending continuously UART data, the Bluetooth module after some seconds would become unresponsive, not being able to perform any further actions such as send/receiving data and disconnecting/connecting. The values shown in Table 5.4 are referent to a 10 second test.

With this module, a reliable connection with sufficient throughput was achieved when connected to a PC, providing a suitable alternative for BM78. The overall performance of this module, if using a PC, is more than required to achieve throughput objectives, supporting up to 600 kb/s using the Serial Port Profile and consequently sending data resorting to RFCOMM. Throughput would possibly be even higher if using directly an L2CAP connection oriented channel to convey application data.

However, after successful tests using a PC, the respective documentation regarding using iAP with the WT12, it was discovered that this solution would not be possible, as Apple's protocol introduces severe throughput restraints to communications, and would never allow for the requirements to be achieved. Therefore discarding this module as a solution for this problem, albeit it's far superior performance when compared with the BM78.

5.2 Bluetooth Low Energy

In order evaluate the proposed BLE solution two different modules were chosen, in order to also provide a power consumption and performance comparison. The modules tested were Espressif's ESP32, and Texas Intruments CC2640R2, using common development boards, ESP32-WROOM and CC2640R2LAUNCHXL, respectively. Code was developed for each module, in order to provide a UART tunnel from Atmel's SAM microcontroller and an iPad, simulating Adapttech's current tInterface system. Tests were also performed with two different versions of the iPad, a 2017 iPad Air 2 and an iPad 2018 were used. In order to capture traffic and characterize all of the Bluetooth devices involved an Ubertooth One platform was used. A brief description of the generic system used in BLE tests is shown in Fig. 5.21.

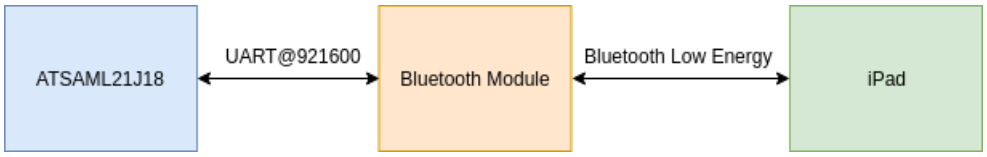

Figure 5.21: System Diagram

As using an Apple iPad is a requirement for this system, all further tests were performed accordingly, as there is no extra hardware required in order to communicate with Apple's equipment using BLE. However two different iPads were used. This is because although both devices are advertised as having Bluetooth v4.2 compliant hardware, their supported features are different, as the older iPad Air 2 does not support Data Length Extension and a few other features, seen in Fig. 5.22. The maximum supported ATT MTU for the iPad Air 2 is also 185 bytes, whilst the newer iPad 2018 has shown support up to 527 bytes from data collected during tests.

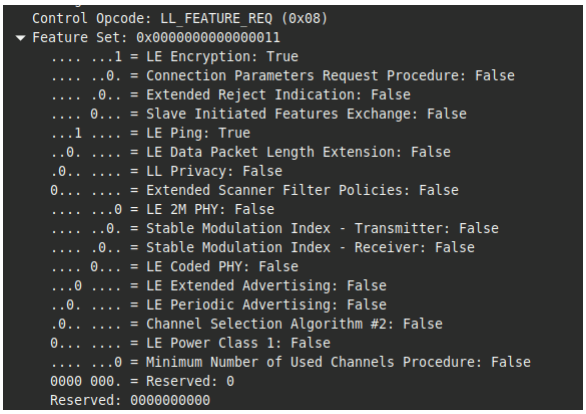

Figure 5.22: Ipad Air 2 Features

As GATT data transfers using notifications imposes less overhead and latency, i.e. only 3 bytes and requiring no response, a characteristic with the notifying property will be used to transfer data. As such, the ATT MTU is an important parameter in order to reduce overhead as it directly affects communication overhead. The best case for this specific application is an L2CAP MTU bigger than the maximum length of an application message, 177 bytes, plus 3 bytes counting with the ATT header. This is because if the ATT MTU is smaller than the maximum message of the application, additional notifications will have to be sent in order to transfer the entire application message.

With the iPad Air 2 only supporting the default 27 byte Link Layer payload size, bigger L2CAP PDUs will consequently suffer segmentation at the Link Layer as illustrated in Fig. 5.23

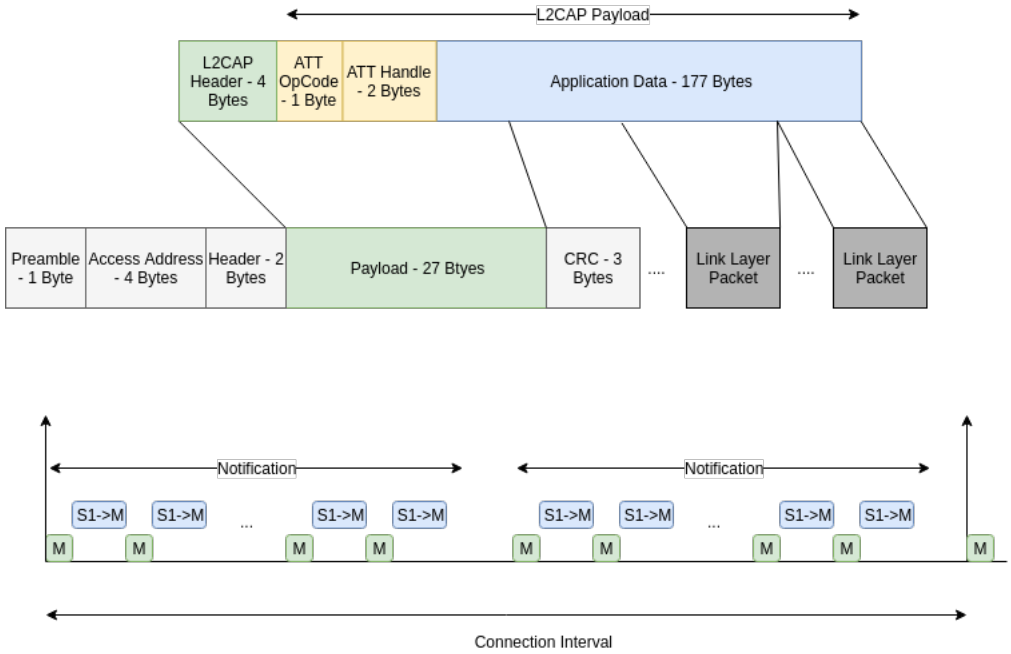

Figure 5.23: Application Data segmentation

With encapsulation, in order to transfer 177 bytes of application data, an L2CAP PDU with 184 bytes will be submitted to the Link Layer. 7 packets will have to be sent from slave to master

although the last packet has a shorter length.

The calculation of necessary LL packets:

$$N_p = \left\lceil \frac{184}{27} \right\rceil = 7$$

A calculation of the required transmission of time for a notification can be defined as:

$$6 \times (80 + 150 + 328 + 150)\mu s + (80 + 150 + 240 + 150)\mu s = 4868\mu s$$

The 150 μs parcels are related to the interframe space required in between each packet sent either by the master or the slave. The other parcels are the respective transmission times for the packets sent. The 80 μs is the time required to transmit a NULL packet, the 328 μs is the time required for the transmission of a message with 27 payload bytes and the 240 μs for the remaining packet.

Using a connection interval of 15ms, the lowest value the iPad accepts, up to 3 notifications could be sent per connection event, enabling a maximum sampling frequency for the tInterface of 200 Hz, a packet diagram is shown in Fig. 5.23, illustrating how a notification is sent.

$$Tp = \frac{Notifications \times Data}{ConnectionInterval}$$

$$Tp = \frac{3 \times 177 \times 8}{15ms} = 283,2kb/s$$

For the iPad 2018, as DLE is supported, maximum application throughput is obviously different. The transmission time for a notification is much smaller, as only a single packet is required to send a complete application message:

$$T_n = (80 + 150 + 1552 + 150)\mu s = 1932\mu s$$

This would allow, in a best case scenario, for 7 ATT notifications to delivered to the iPad. Maximum throughput can be calculated as:

$$TpDLE = \frac{7 \times 177 \times 8}{15ms} = 660,8kb/s$$

5.2.1 ESP32

Using example code provided with ESP-IDF, the development framework provided by the respective manufacturer, adaptations were made in order to be able to configure Apple's recommended 15 ms connection interval and 0 slave latency. The primary example used, ***spp server*** , is a GATT server composed of one service, with 3 characteristics. A packet parser was also added to the example code, in order to identify application messages and submit them unsegmented to the GATT server. The UART driver on the ESP32 was also altered to use a baud rate 921600 bps, in order to reduce UART communication time.

When connecting, both devices a perform series of procedures. As observed in Fig. 5.24, initiation of connections happens when the initiator device (iPad) sends a request to the ESP32, CONNECT_REQ with the required parameters in order to setup a connection.

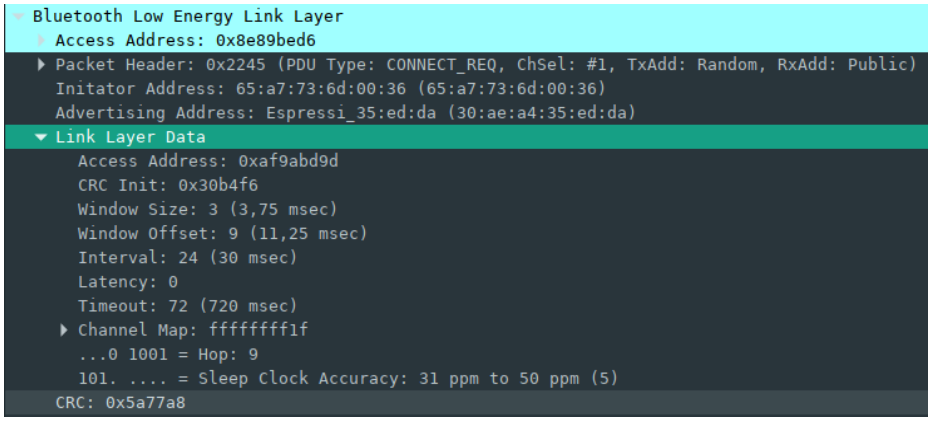

Figure 5.24: Packet Capture Timestamps

After this Link Layer versions are exchanged between devices with the two devices already having formed a piconet. In parallel the ESP32 sends a LENGTH_REQ, in order to request a Link Layer payload size of 251 bytes, however the response from the iPad Air 2 is a LL_UNKNOWN_REQ keeping the connection with a default maximum Link Layer Payload of 27 bytes, as seen in Fig. 5.25.

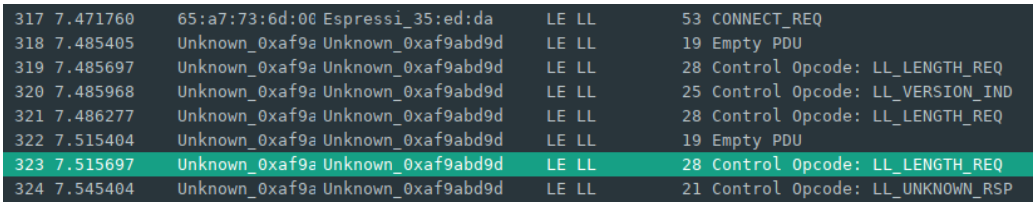

Figure 5.25: LL_LENGHT_REQ Failure

With the other iPad tablet, the response confirmed the maximum Link Layer payload size increase to 251 bytes. The following procedures performed include a feature exchange, an update on connection parameters to a connection interval of 15ms(Fig. 5.26) and a L2CAP MTU exchange between both devices (Fig. 5.27).

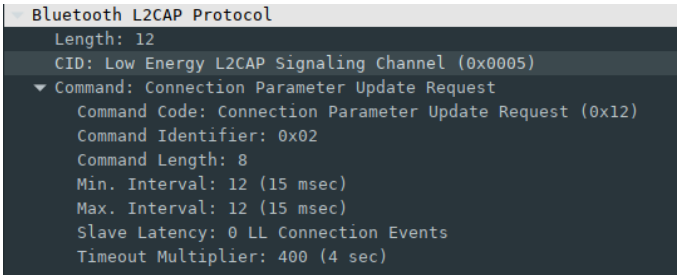

Figure 5.26: Link Layer Parameter Update

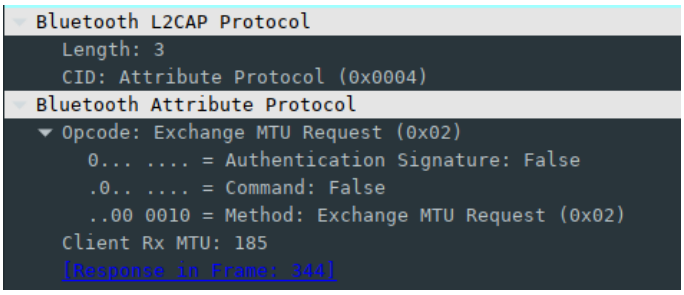

Figure 5.27: MTU Exchange

Logs from the captures performed during the test with the Ubertooth show that this platform was able to record most of the interaction between the iPad and ESP32 devices. However, an erratic behaviour on Wireshark's timestamps could be seen, skipping several milliseconds at a time as shown in Fig. 5.28. In a detailed inspection, it was observed that messages received using the LightBlue iOS app were missing from the Ubertooth packet log. As this application did not provide any suitable data about connection throughput and the Ubertooth wasn't able to capture every packet in the connection, an iOS application with a simple interface was developed in order to properly evaluate throughput. The application was created, based of Adafruit's *Basic Chat* open-source application written in Swift3 (Fig .5.29). Throughput values shown in the following table have been calculated resorting to this application.

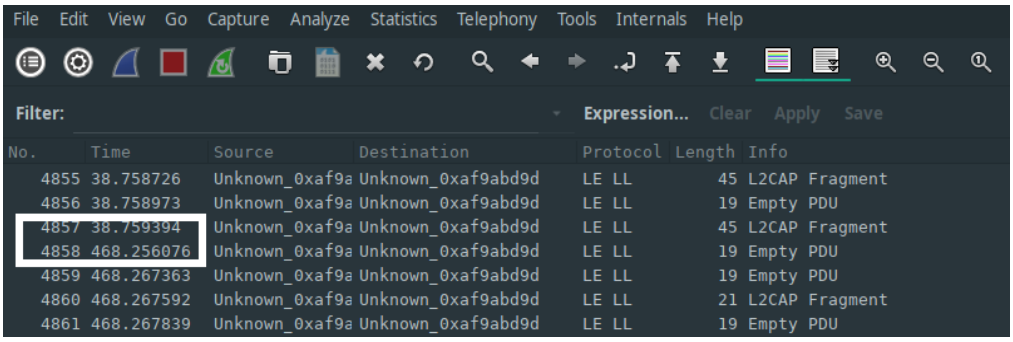

Figure 5.28: Packet Capture Timestamps

As shown in Table 5.5, the throughput values achieved can be seen relatively to the system's sampling period. This test was performed with a connection interval of 15ms and a slave latency of 0ms and for both iPads, with a duration of 1 minute.

Period	Required Throughput	iPad Air 2 (Avg)	iPad Air 2 (Max)	iPad 2018 (Avg)	iPad 2018 (Max)
20 ms	70,800 kb/s	70,800 kb/s	71,200 kb/s	68,200 kb/s	70,8 kb/s
15 ms	94,3056 kb/s	94,3056 kb/s	95,410 kb/s	90,88 kb/s	96,288 kb/s
10 ms	141,600 kb/s	94,3056 kb/s	95,410 kb/s	133,53 kb/s	143,016 kb/s
5 ms	283200 bps	N/A	N/A	N/A	N/A

Table 5.5: Achieved Results

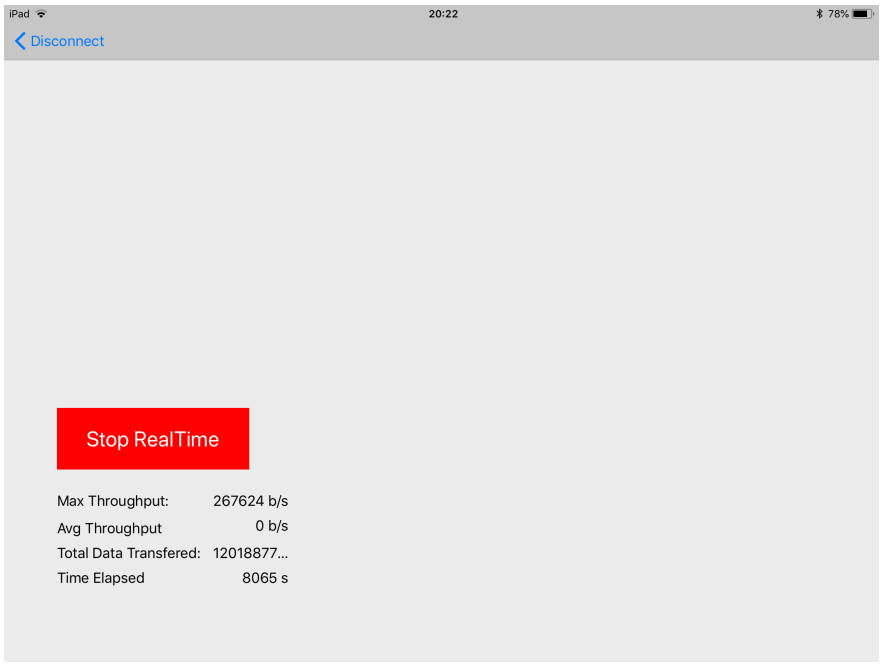

Figure 5.29: iOS Test application

In the following graphs, the throughput data is shown for the different tablets and sampling periods. As it was not of interest, within the topic of this dissertation, to make a graphic in the application itself, data shown is plotted from data logged in XCode's console, the Integrated Development Environment (IDE) used for development.

- Throughput Graph, test period 20ms (Bits/second):

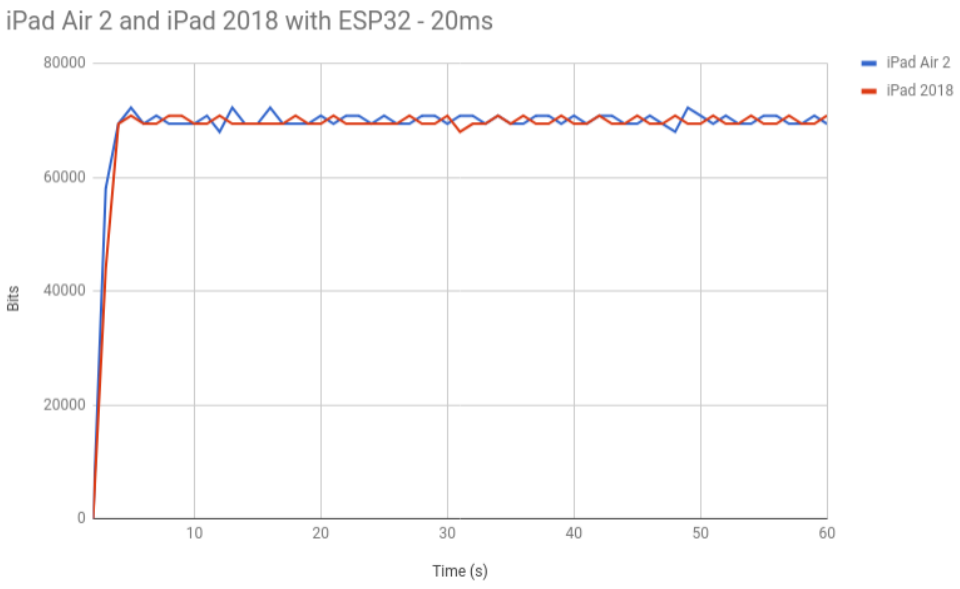

Figure 5.30: ESP32 Throughput test - 20 ms

- Throughput Graph, test period 15ms (Bits/second):

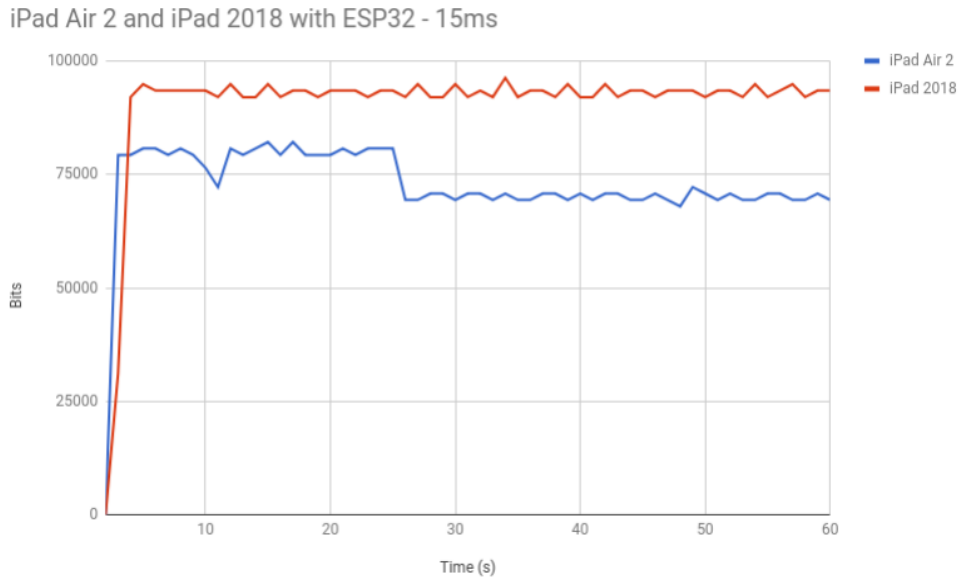

Figure 5.31: ESP32 Throughput test - 15 ms

- Throughput Graph, test period 10ms (Bits/second):

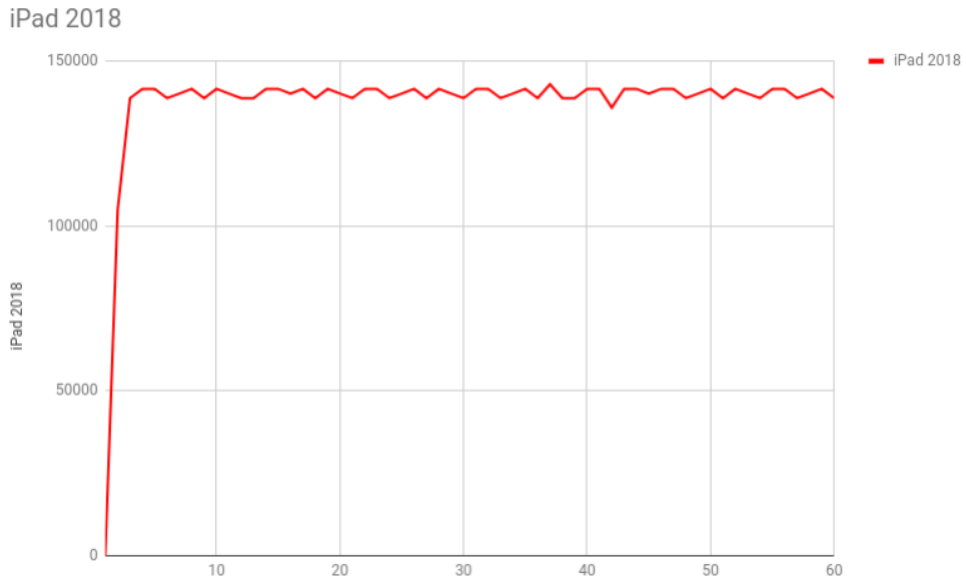

Figure 5.32: ESP32 Throughput test - 10 ms

- Throughput Graph, test period 5ms (Bits/second):

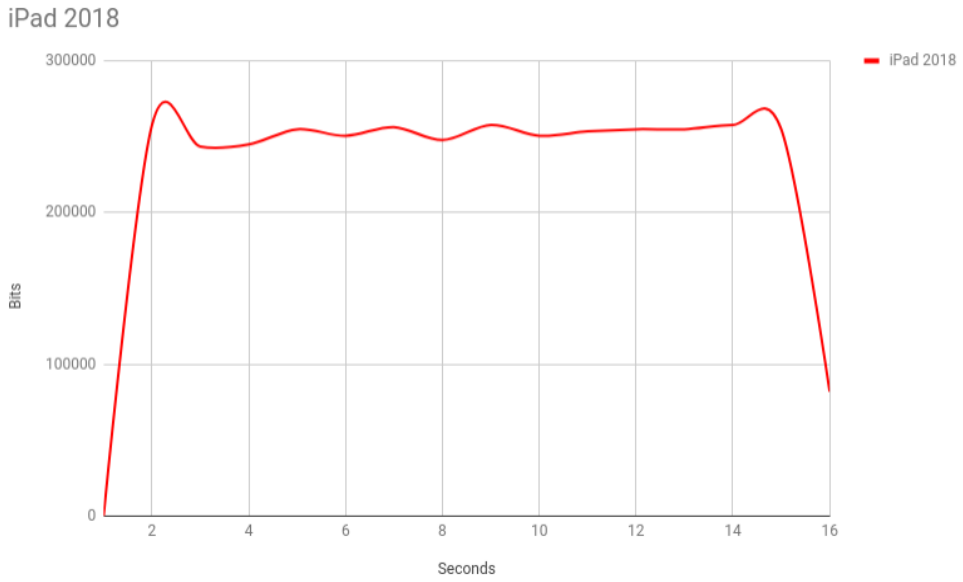

Figure 5.33: ESP32 Throughput test - 5 ms

As seen in the figures and respective tables, there is a discrepancy between the required throughput and the measured. This might happen because of unknown characteristics in the iOS app scheduling, as there is a periodic 1 second task measuring throughput. Although this happen, it was possible to verify that in cases where the difference is small, 2 or 3 kb/s, no messages were lost, and therefore the reason as to why that happens remains unknown.

While performing some tests with the iPad Air 2 the ESP32 would often freeze in more demanding situations. As such, there is no information in regarding this tablet for the 10ms and 5ms tests.

Measured using the iPad application developed for this purpose, the maximum achieved average throughput using an iPad Air 2 was 94 kb/s . However, this value is much lower than what is theoretically achievable, as often specific Bluetooth stacks impose a limit on the number of packets and/or notifications per connection event, due to implementation. Another cause for this might also be related with the code produced and with task concurrency in the ESP32.

In the traffic analysis, it can be noticed that notifications are divided into 8 L2CAP fragments, when communicating with the iPad Air 2. As the master, the iPad controls the piconet. By sending packets to a slave during a connection event, it enables the slave to respond subsequently, transferring data from slave to master. If no packets are sent by the master, the slave isn't permitted to transmit packets.

From the traffic analysis performed, it can be seen that each notification is divided into 8 L2CAP fragments, 7 with a LL payload of 26 and a remaining packet with 2 bytes of payload, totaling the 184 bytes of the original L2CAP PDU. Observing Ubertooth packet logs, it can be seen that the first packets are quickly transmitted, as the master is rapidly sending packets and enabling the ESP32 to transmit data. However, normally after 6 packets are sent, the master stays

silent and only returns to communicate with the slave in the end of the connection event (Fig. 5.34). However, often there are discrepancies in timestamps provided in the packet, showing some lack of reliability of the Ubertooth for higher data rate connections.

4730	*REF*	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment Start
4731	0.000247	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4732	0.000668	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4733	0.000915	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4734	0.001336	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4735	0.001583	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4736	0.002004	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4737	0.002251	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4738	0.002672	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4739	0.002919	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4740	0.003349	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4741	0.014103	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4742	0.014532	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	45 L2CAP Fragment
4743	0.014772	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	LE LL	19 Empty PDU
4744	0.015000	Unknown_0xaf9abd9d	Unknown_0xaf9abd9d	ATT	21 UnknownDirection Handle Value Notification, Handle: 0x002c (Unknown: Unknown)

Figure 5.34: Packet Capture with iPad Air 2

With the iPad version released in 2018, supporting Data Length Extension, reliable communication was achieved with a sampling frequency of 100Hz and a throughput of 141kb/s. A maximum throughput of 263,4 kb/s also was achieved using a sampling frequency of 200 Hz, although many application messages were not received and therefore does not permit the *tInterface* device to reliably transmit application data at this rate. As the Ubertooth firmware does not appear to be able to capture Link Layer packets with more than 41 bytes, and only NULL packets were captured when attempting to monitor traffic, no further traffic analysis was performed for this tablet.

5.2.2 Texas Instruments CC2640R2

The ESP32's development environment were found to be still in development, and not all BLE features available in the hardware are implemented at a firmware level. Also, as support material was found to be less than expected, a search was performed to find a better supported platform. The platform chosen for further tests was TI's CC2640R2.

Two types of tests were performed with this module, first connecting to an iPad and testing throughput with Bluetooth v4.2 with and without DLE such as for the ESP32.

The setup of the tests is the same as for the previous module, and 5 different tests periods were used. The code produced was based in a example provided with the default SDK, *spp_ble_server*, and several modifications were made in order to enhance its application data transmission rate. In the last test performed, the module would start to behave erratically after some seconds, not being able to recover. This might be a cause of implementation, or hardware constraints such as memory.

As the connection setup is similar to the one with the ESP32, it will not be discussed.

The following table summarizes the throughput values obtained when testing. A 15ms connection interval and 0 slave latency were the used parameters in the connection.

Period	Required Throughput	iPad Air 2 (Avg)	iPad Air 2 (Max)	iPad 2018 (Avg)	iPad 2018 (Max)
20 ms	70,800 kb/s	70,800 kb/s	71,200 kb/s	68,91 kb/s	72,216 kb/s
15 ms	943,056 kb/s	94,3056 kb/s	95,410 kb/s	91,12 kb/s	94,872 kb/s
10 ms	141,600 kb/s	94,3056 kb/s	95,410 kb/s	135,16 kb/s	144,432 kb/s
5 ms	283,200 kb/s	N/A	N/A	244,75 kb/s	267,624 kb/s
2,5 ms	566,400 kb/s	N/A	N/A	N/A	N/A

Table 5.6: Achieved Results

- Throughput Graph, test period 20ms (Bits/second):

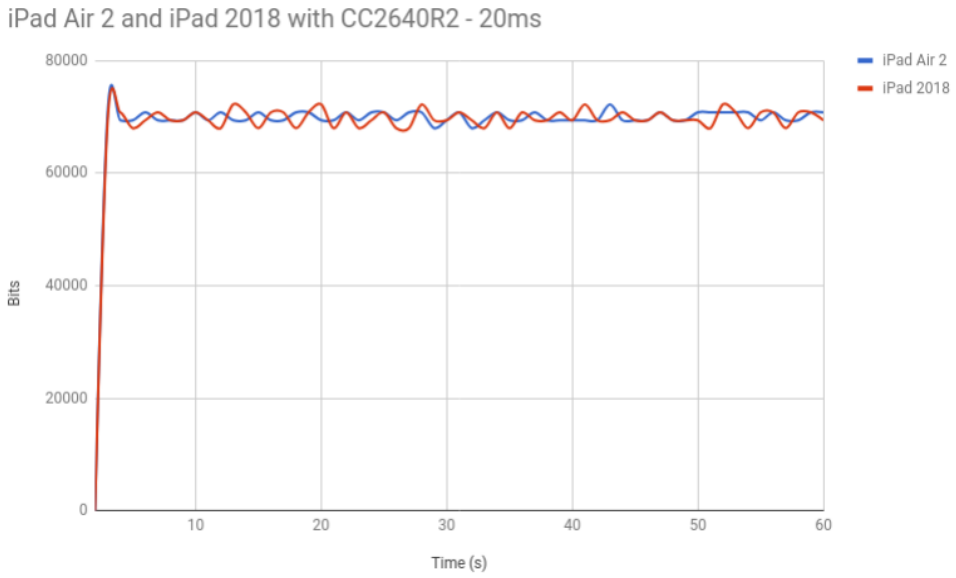

Figure 5.35: CC2640R2 Throughput test - 20 ms

- Throughput Graph, test period 15ms (Bits/second):

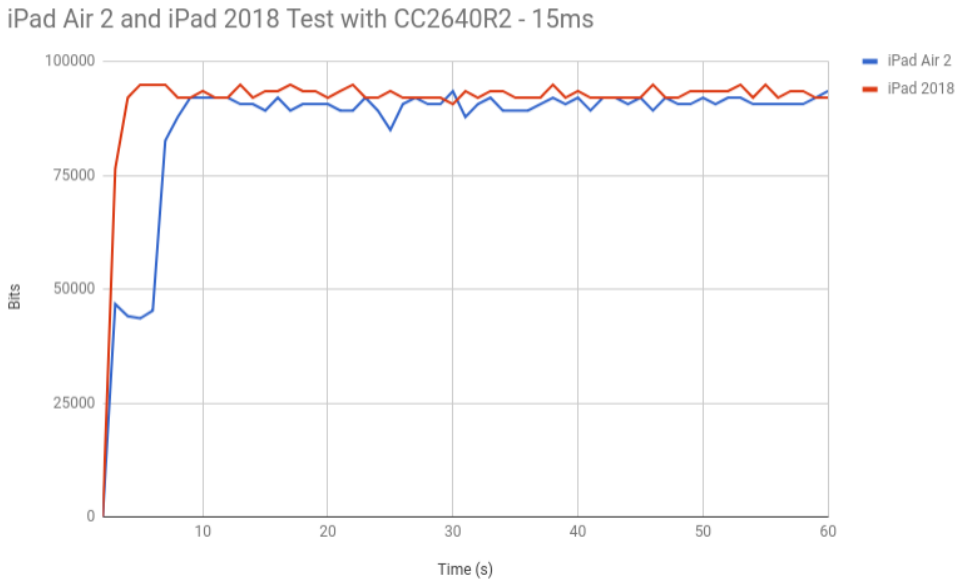

Figure 5.36: CC2640R2 Throughput test - 15 ms

- Throughput Graph, test period 10ms (Bits/second):

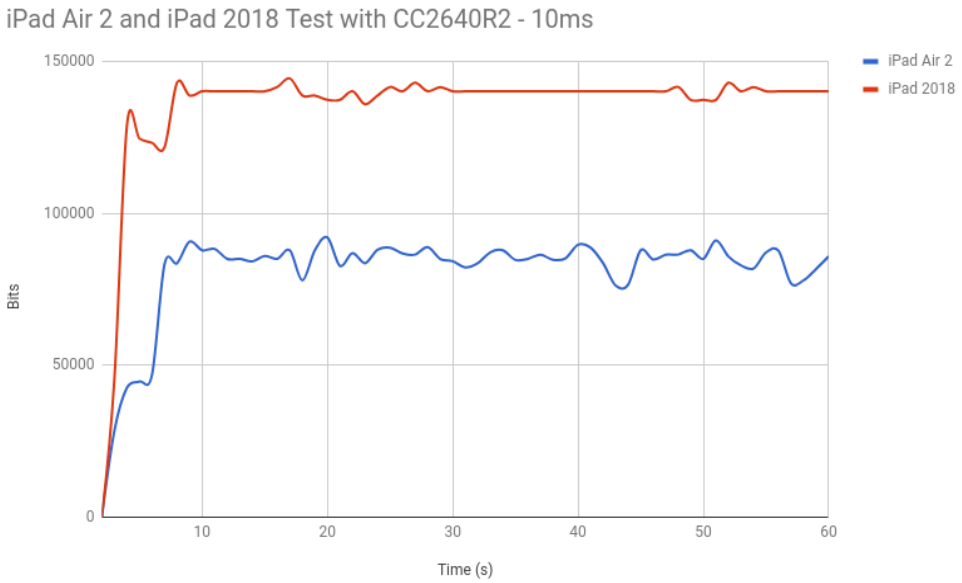

Figure 5.37: CC2640R2 Throughput test - 10 ms

- Throughput Graph, test period 5ms (Bits/second):

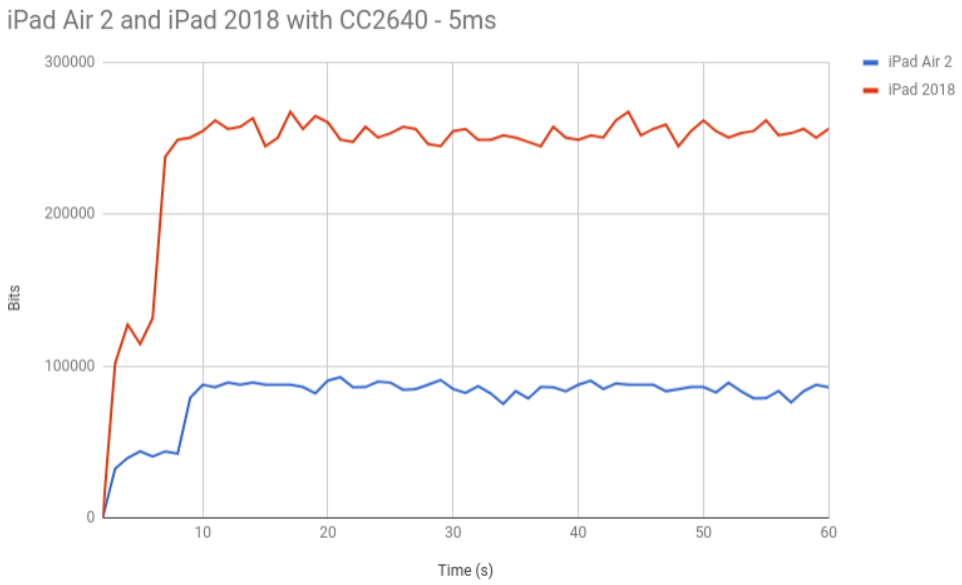

Figure 5.38: CC2640R2 Throughput test - 5 ms

- Throughput Graph, test period 2,5ms (Bits/second):

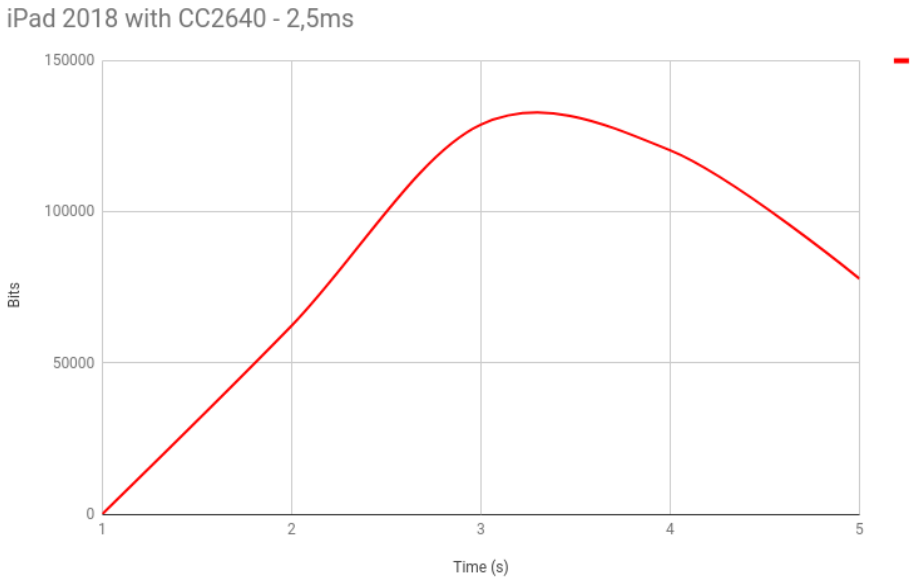

Figure 5.39: CC2640R2 Throughput test - 2,5 ms

The tests performed revealed a superior performance with this module, as the CC2640R2 was able to fulfill throughput requirements in the 20ms and 15ms tests for both iPads. When using the iPad 2018 in the 5ms test, the CC2640R2 was able to achieve a maximum throughput of 267,624 kb/s, very close to the required throughput of 283,200 kb/s. The existing gap between the required and

measured throughput values occur because the available L2CAP buffer is full, as not all messages submitted sent immediately.

When using a period of 2,5 ms for application messages, communication was not possible, as the connection halted some seconds after its start. This is not related to the Bluetooth protocol, but with the hardware itself or code implementation. However, as the tests show, a maximum throughput of 267;62 kb/s was achieved using the CC2640R2, making this *SoC* a viable solution for the system's throughput problems.

5.3 WiFi

As the ESP32 already has peripherals supporting WiFi, a test was also performed in order to evaluate this module's performance using another wireless technology.

After some research, a network bandwidth analysis tool, *iPerf*, was found and used in order to test this *SoC's* performance using TCP sockets. In the microcontroller, an example provided with the default framework, ESP-IDF, was used in order to send data to the TCP server running in a desktop PC. A Hauwei HG8247H router was used as a WiFi Access Point.

The output on both programs consoles can be seen in Figs. 5.40 and 5.41.

```
iperf -c 192.168.1.236 -i 3 -t 60
I (249383) iperf: mode=tcp-client sip=192.168.1.243:5001, dip=192.168.1.236:5001, interval=3, time=60
esp32>
        Interval Bandwidth
   0-   3 sec       5.16 Mbits/sec
   3-   6 sec       6.99 Mbits/sec
   6-   9 sec       9.52 Mbits/sec
   9-  12 sec       9.96 Mbits/sec
  12-  15 sec       10.92 Mbits/sec
  15-  18 sec       11.10 Mbits/sec
  18-  21 sec       9.61 Mbits/sec
  21-  24 sec       13.81 Mbits/sec
  24-  27 sec       14.77 Mbits/sec
  27-  30 sec       17.83 Mbits/sec
  30-  33 sec       18.44 Mbits/sec
  33-  36 sec       18.18 Mbits/sec
  36-  39 sec       18.35 Mbits/sec
  39-  42 sec       18.52 Mbits/sec
  42-  45 sec       15.99 Mbits/sec
  45-  48 sec       15.82 Mbits/sec
  48-  51 sec       17.83 Mbits/sec
  51-  54 sec       19.66 Mbits/sec
  54-  57 sec       19.31 Mbits/sec
  57-  60 sec       19.57 Mbits/sec
   0-  60 sec       14.60 Mbits/sec
I (309484) iperf: iperf exit
```

Figure 5.40: ESP32 iPerf output

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.236 port 5001 connected with 192.168.1.243 port 56387 (peer 0.0.51354-unk)
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 3.0 sec  1.87 MBytes  5.22 Mbits/sec
[  4]  3.0- 6.0 sec  2.49 MBytes  6.97 Mbits/sec
[  4]  6.0- 9.0 sec  3.34 MBytes  9.34 Mbits/sec
[  4]  9.0-12.0 sec  3.62 MBytes  10.1 Mbits/sec
[  4] 12.0-15.0 sec  3.90 MBytes  10.9 Mbits/sec
[  4] 15.0-18.0 sec  3.97 MBytes  11.1 Mbits/sec
[  4] 18.0-21.0 sec  3.48 MBytes  9.74 Mbits/sec
[  4] 21.0-24.0 sec  4.95 MBytes  13.8 Mbits/sec
[  4] 24.0-27.0 sec  5.26 MBytes  14.7 Mbits/sec
[  4] 27.0-30.0 sec  6.36 MBytes  17.8 Mbits/sec
[  4] 30.0-33.0 sec  6.60 MBytes  18.5 Mbits/sec
[  4] 33.0-36.0 sec  6.53 MBytes  18.3 Mbits/sec
[  4] 36.0-39.0 sec  6.55 MBytes  18.3 Mbits/sec
[  4] 39.0-42.0 sec  6.61 MBytes  18.5 Mbits/sec
[  4] 42.0-45.0 sec  5.76 MBytes  16.1 Mbits/sec
[  4] 45.0-48.0 sec  5.68 MBytes  15.9 Mbits/sec
[  4] 48.0-51.0 sec  6.36 MBytes  17.8 Mbits/sec
[  4] 51.0-54.0 sec  7.04 MBytes  19.7 Mbits/sec
[  4] 54.0-57.0 sec  6.88 MBytes  19.2 Mbits/sec
[  4] 57.0-60.0 sec  7.00 MBytes  19.6 Mbits/sec
[  4]  0.0-60.1 sec   104 MBytes  14.6 Mbits/sec
```

Figure 5.41: PC iPerf output

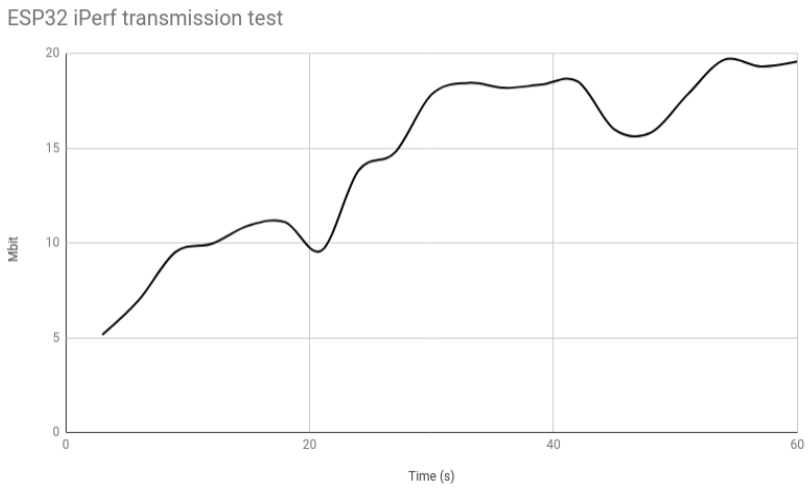

Figure 5.42: ESP32 Wifi throughput graph

The test had a duration of one minute, achieving a maximum throughput of 19,66 mb/s and an average of 14,56 mb/s (Fig 5.42). As expected, a much higher throughput was achieved using the ESP32 and WiFi, as baseband data rates are on another scale than that of Bluetooth.

5.4 Power Consumption

As the *tInterface* is a battery powered wearable device, the power consumption is also an important factor to take into account when considering different wireless technologies and systems. The modules/*SoCs* that were analyzed previously, apart from the BM78, were also tested in terms of power consumption.

Using an Hantek 365 F data-logger/multimeter (Fig. 5.43), data relative to the average current which is being consumed by each module could be determined. An important note is that, although tests were performed using development kits and breakout boards, the consumption may be higher, as some of the boards used do not provide an interface to measure only the current drawn by the modules/*SoCs*, but the whole system, which may include other components that have influence on the measurements.

Figure 5.43: Hantek 365 F

Tests for power consumption were performed for the 3 devices, when not connected but in an advertising/waiting state, when connected, and in real-time operation. All modules were evaluated using an application sampling period of 10ms, in order to provide an equal comparison base.

Module/SoC	In Standy-by	Connected	In Operation
WT12	85,8 mW	155,1 mW	201,3 mW
ESP32 (BLE)	369.6 mW	385,4 mW	405,9 mW
ESP32 (WiFi)	608.5 mW	790,0 mW	950,2 mW
CC2640R2	6,435 mW	41,25 mW	49,5 mW

Table 5.7: Power Consumption Test Results

As it can be seen, the device that consumes overall less power if by far the TI CC2640R2, followed by the BM78 and then the ESP32. Even if using BLE with the ESP32, power consumption is still much higher than of its counterparts, which might be due to the numerous array of peripherals present in the ESP32, it's clock rate of up to 240MHz and the additional ICs that can be found in the development board used. When comparing both tests performed with the ESP32, a significantly higher consumption occurred when using WiFi, although power consumption using

BLE is also relatively high. The CC2640R2, even when operating at a relatively high throughput, consumes very little power, providing a very attractive solution.

5.5 Summary

The results obtained when testing the BM78 module, point out that it does not perform in a way that is viable for the product's requirements. Its behaviour is not stable, as it performs differently when repeating the same tests in the same conditions. This is the reason why an overview of the Bluetooth protocol was required, and a search for other modules and *SoCs*. Also, as RFCOMM DLCs rely on lower layers for error correction, segmentation and reassembly, and the BM78 fails to provide reliability, performance is reduced.

None of these mechanisms are provided by RFCOMM and, furthermore, the UIH frames are supposed to be employed in cases were the data integrity is less important than its actual delivery, as stated in TS 07.10 . With these characteristics it is only natural that occasional errors appear in communications with this module. Even more, as the implementation of actual LMP and L2CAP managers in the BM78 is unknown and of closed source, the causes for missing/incomplete messages are very hard to pinpoint. Given this, a conclusion is reached that the BM78 is not an appropriate solution for this specific application.

> "UIH is used where the integrity of the information being transferred is of lesser importance than its delivery to the correct DLCI. For the UIH frame, the FCS shall be calculated over only the address, control and length fields." [4]

Using the WT12 Bluetooth Classic module connected to a PC, tests have shown that Bluetooth Classic is still a viable candidate for use in high data rate PANs. However, when using iAP and Apple hardware, various constraints arise, affecting further development. As it is an imperative requirement to use an iPad to communicate with the *tInterface*, BLE presented a more appealing perspective as implementation does not require additional hardware or entering any specific Apple program such as MFi.

With the tests performed the ESP32 and TI's CC2640R2, the throughput achieved was somehow similar and, although the CC2640R2 had better overall performance and power consumption, requirements were achieved satisfactory with both Bluetooth modules. After implementation and testing this solution was validated as being able to provide a real time message each 20ms or 50 Hz and also 15ms, or 66,7Hz whilst testing with the iPad Air 2. When using the Data Length Extension feature with the iPad 2018, the maximum allowed sampling frequency for the system was extended to 100Hz, allowing a significant performance increase. It is important to notice that the maximum throughput for the application is not equal to the link's throughput, as the application is required to use a set of predetermined sampling frequencies, as required by the gait analysis algorithms used. The reason of the difference between the theoretical and practical maximum throughput values with Bluetooth Low Energy can be directly linked to the iPad, as it assumes the master role in the connection and controls the timming of transmission slots for slaves.

Results from the tests conducted with the ESP32 module using WiFi have shown that it is much easier to achieve higher throughput whilst using this technology. The required throughput was achieved using TCP sockets, although with UDP results could possibly be higher, this way the use of a reliable connection based communication protocol already delivers application data unsegmented and in order. However, power consumption is much higher than when using BR/EDR or BLE, which would be a serious disadvantage for the product, if implemented.

5.5.1 Ubertooth

The Ubertooth One is an open-source and open-hardware device enabling Bluetooth traffic analysis, packet injection and a few other features. The firmware provided by the project enables the user to promptly use the device with packet analyzing software such as Wireshark or Kismet. For BLE traffic analysis, the ubertooth-btle tool starts by listening to advertising packets, and follows device connections upon a connection request. After capturing packets, these are transmitted via a USB Virtual Serial Port to a driver on the host PC which then dumps them to a FIFO queue. The traffic analysis software then reads the FIFO file and processes packets, enabling the user to graphically interpret and decode communications.

As the development of this hardware is not directly paid to its developers, and although the platform behave extremely well for its price tag, some limitations were found when using the device to monitor throughput. Evidently some packets were missed, either because of queuing or enviromental factors, that could be seen delivered to the application but not found in the capture log. This provoked further research in order to clarify if it was possible to, with this hardware, capture and evaluate effectively LE connections throughput. Another limitation of this hardware is that for a 2Mb/s PHY layer such has the one present in BT v2.1, there is no possibility of sniffing, as the radio front end present in the Ubertooth One (Texas Instruments CC2400) does not enable for demodulation of 4GFSK. This limits the Ubertooth sniffing to Bluetooth LE until v4.2 and Basic Rate connections only.

5.5.2 Other Considerations

> "ESP32 integrates Wi-Fi (2.4 GHz band) and Bluetooth 4.2 solutions on a single chip, along with dual high performance cores, Ultra Low Power co-processor and several peripherals. Powered by 40 nm technology, ESP32 provides a robust, highly integrated platform to meet the continuous demands for efficient power usage, compact design, security, high performance, and reliability."

This microcontroller is produced by Espressif, being packed with features such as Wi-Fi, Bluetooth Dual Mode (LE and BR/EDR), and supported by the ESP-IDF (IoT Development Framework). Implementation of BLE 4.2 firmware to substitute the previous module required some reading, configuration and research in order to achieve this solution. The ESP-IDF is a framework which makes use of a specific FreeRTOS port for the ESP32, a widely supported and popular RTOS, managing all of the task processing, scheduling and execution on the main CPU. Being a

fairly simple and intuitive real time operating system, most of the time was applied to understanding the Bluedroid Stack, the default stack used by ESP-IDF and Android v4.2+ platforms, and the API provided to interact with it. Although other Bluetooth stacks may be ported and adapted to use with the ESP32, such as the BTstack developed by Swiss company BlueKitchen, no further research was performed. As examples provided are not feature rich, some work was carried out in order to implement GAP and GATT services, resorting to various examples.

The market availability of this hardware is increasingly bigger, as the ESP32 becomes a more widespread microcontroller. However large order purchases are not available in semiconductor distributors such as Mouser Electronics or Farnell, thus being a decisive factor when adopting this hardware for mass production product. Although every tool tested is correctly working, Espressif proves to have a specific methodology in developing embedded solutions, firstly developing the hardware, launching it into the market and developing the framework while already on the market. With this approach some problems arise, such as security implementation in BLE devices, which doesn't fully support every pairing mechanism. Some of these particularities make the ESP32 an attractive solution for easy development, but not appropriate for every kind of product, especially for products/companies requiring extensive support from manufacturers.

From the experience gained developing a BLE solution with the CC2640R2, it can be said that TI provides engineers with many different resources. Extensive support, guides and tutorials are easily available and a fully configured IDE is also provided. The *SoC* itself is sold in discrete units without an antenna or crystal oscillators, however different manufactures produce modules including this IC such as Laird's Sable-X-R2 module, easily accessible. Texas Instruments also provides its own RTOS, the TI-RTOS, which enables for a good environment in order to develop embedded solutions, an array of drivers and code examples are also available and well documented, making this an attractive platform.

One great feature about the CC2640R2 is that it is composed of several processing units, and its RF core is able to support different radio specifications such as ZigBee, Bluetooth or other proprietary protocols. Also, TI already offers a proprietary pre-certified Bluetooth 5 stack, enabling products were this hardware is present, to use the latest version of the Bluetooth Specification and benefit of its advantages.

Chapter 6

Wireless Inertial Motion Unit

After achieving initial requirements, a secondary requirement for the practical work developed in this dissertation was taken into account, which was to develop a second IMU (Inertial Motion Unit) system communicating wirelessly with the main system (Wearable/iPad). The purpose of this second IMU is to relay more data about lower limb movement in order to develop machine learning algorithms to characterize and analyze a patient's gait. The current solution employed is to have a cable connected from the wearable device into the second IMU (Fig. 6.1)which has some disadvantages in terms of usability.

Figure 6.1: *tInterface*

This secondary requirement aims for the replacement of this cable, effectively creating another, independent, device. The solution, regardless of technology implementation, should be as transparent to the user as possible and have the least amount of impact in usability of the product as possible.

Taking this into consideration an analysis into this other requirement was performed, in order to open a path to future work around this topic.

6.1 Analysis

Prior to any implementation, a review of possible solutions was performed, in order to produce a device that meets all of the defined requirements. First a qualitative analysis of each technology will be performed, regarding several key parameters and, secondly, a technical revision of the solution to implement will be performed.

6.1.1 Requirements

In order to add this new device into the system, several requirements have to be met:

- To support a payload size of 22 bytes, 18 for IMU data and a 4 byte timestamp.
- To be able to have a battery duration and sampling frequency equal or greater than the *tInterface*.
- Have a small form factor and to be lightweight.

6.1.2 Technology

There are various technological solutions that could be used in this product in order to satisfy functional requirements, such as Bluetooth, Zigbee, ANT or even other proprietary radio specification. In order to evaluate which technology fits best this product, some facts have to be taken into account. The implementation and time to market are a recurring theme in discussion on a corporate environment, as such, choosing a technology is not only a matter of technical importance, as it impacts also the company's core business.

A qualitative table is presented next, in order to better understand possible advantages and disadvantages of each technological solution.

Technology	Power Consumption	Throughput	Supported by iPad	Implementation Difficulty in current System / Reason
BLE	Low	<1mb/s	Yes	Easy - Already supported in current version
BR/EDR	Medium	>1mb/s	Yes	Medium - Requires extra hardware
ZigBee	Low	<250kb/s	No	Difficult - Requires new Hardware and Knowledge Base
ANT	Low	<60kb/s	Yes, with external module	Difficult - Requires new Hardware and Knowledge Base
WiFi	High	>10mb/s	Yes	Easy - Already supported in future version

Table 6.1: Wireless technology comparison

All of the studied technologies meet throughput requirements, this will not be the defining parameter in order to choose a wireless technology for this device. Choosing a technology that is supported by the iPad is very important, although not imperative, as long as data can be relayed through another system which can communicate with the tablet.

ZigBee is not a viable candidate a it's not supported by Apple's iPad, making it a last resort as a choice, despite being an accessible and widespread technology. In order to use ZigBee in a system, one would have to have 2 different wireless standards operating within a third system to be able to act as a gateway, which is viable, despite possible concurrency, escalation and throughput problems.

The possibility of using BR/EDR does not represent the best solution for the application considered, as it implies higher power consumption and the use of additional hardware in order to be able to connect to the iPad. For this reason, Classic Bluetooth may not the best technology available for this specific purpose.

ANT and ANT+ require an external adapter in order to provide a compatible radio, posing a great disadvantage in terms of usage, making it unfit for this application.

Another possible candidate would be WiFi, but as previously stated, with WiFi it would be increasingly difficult to construct a low-power device. A possible solution for this problem is to implement deep sleep and send data in batches but less frequently. Still, using WiFi would pose a usability problem in terms of a device's required configuration in order to connect to an access point.

Being a previously studied technology, and already implemented in other products, BLE could possibly be the best technological solution for this problem. As shown in the previous chapter, for a sampling rate of 100Hz, there is still bandwidth left for other application data to be streamed to the iPad.

In the next paragraphs a detailed perspective will be given regarding the ideal connection in order to support both devices, and theoretical validation of the possibility of using different network topologies.

As Apple recommends a 15ms connection interval and 0ms slave latency, if two slaves are connected to an iPad, the iPad's scheduler will have to divide each 15ms interval in two, as no connection events can be skipped due to the slave latency parameter, in order to allow both slaves to transmit and receive data. As this is not publicly accessible information, one can only perform theoretical best case calculations and then proceed to verify it experimentally. In the case of a *fair scheduler* [32], each of the anchor points for each device would be 7,5ms apart from the other, as shown in Fig. 6.2

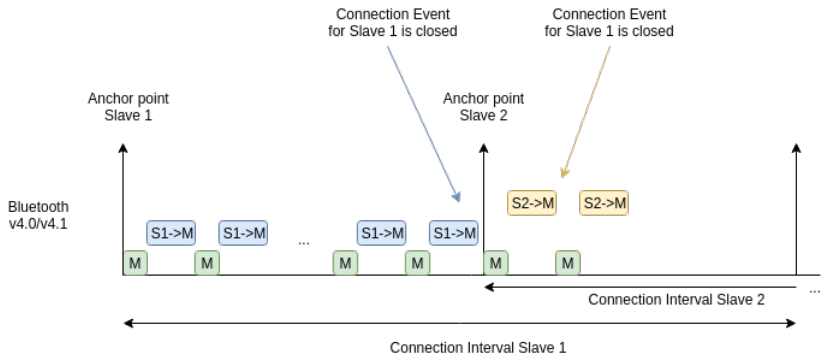

Figure 6.2: Fair Device Scheduling

In this case each slave would have 7.5ms available to communicate with the master, being that the necessary transmission time for each device's data should be lower than this value. Summarizing and dependent on the Bluetooth specification and features used, the necessary time to receive data successfully from both devices is:

- For Bluetooth v4.0/v4.1 with a 27 byte payload, 7 Link Layer packets would be needed to receive the 177 bytes of pressure and inertial sensor data from the *tInterface* device, and 2 packets to receive a 4 byte timestamp and 18 byte IMU data for the Wireless IMU device, totaling 22 bytes. The first packet would contain 20 bytes of payload, resulting in a transmission time of 328 us. For the second packet that would be required, only 2 payload bytes are left to be sent, resulting in 128us of transmission time.

$$6 * (80 + 150 + 328 + 150)\mu s + (80 + 150 + 240 + 150)\mu s = 4868\mu s$$

The total time needed to transmit the 22 bytes of data is defined by:

$$T_2 = (80 + 150 + 328 + 150 + 80 + 150 + 128 + 150)\mu s = 1216$$

$$Total_{4.1} = (5464 + 1216)us = 6680\mu s$$

The total time used by the 2 devices in order to transmit data to the application is 6680us, occupying more than one third of the connection interval. As the tInterface's remaing part of it's connection event is not enough to send another application data message, a maximum sampling period for the device is imposed at 15ms, equivalent to a sampling frequency of 66.6Hz.

- In Bluetooth 4.2, if using DLE, transmission time could be reduced to us, as the tInterface device is able to send it's information much faster as only one packet is needed. For the Wireless IMU device also, only one packet is required to transmit the 22 bytes application data.

$$T_1 = (80 + 150 + 1552 + 150)\mu s = 1932\mu s$$

$$T_2 = (80 + 150 + 344 + 150)\mu s = 724\mu s$$

$$Total_{4.2} = (1932 + 724)\mu s = 2656\mu s$$

As for the tInterface device, the total required transmission time is 1932us, up to three notifications could be sent in ideal conditions. For the other device up to 10 notifications could be sent in 7.5ms. This would permit a maximum sampling frequency for the whole system of 200Hz, with a total throughput of 283,6kb/s.

6.1.3 Possible Network Solutions

As the product currently uses Bluetooth technology, there is a certain advantage in implementing this new IMU system with the same technology. If so, two network topologies could be employed. If connected to the iPad separately both wearable devices would form a piconet with the master device, enabling for an easy implementation regarding the iPad application. With this topology it would even be fairly easy to add more than one Wireless IMU device. There is one disadvantage to this topology which is the fact that the user would have to select each wearable device in the application in order to connect, as with the previous version only one device existed.

The other possible solution would be to have the primary wearable device (*tInterface*) as a middle-man, in which a wireless IMU would send it's data to be relayed through the wearable device and then delivered to the iPad. This would have the advantage of being transparent to the user, as one would only have to select the primary device to connect.

Both topologies are represented in Fig. 6.3.

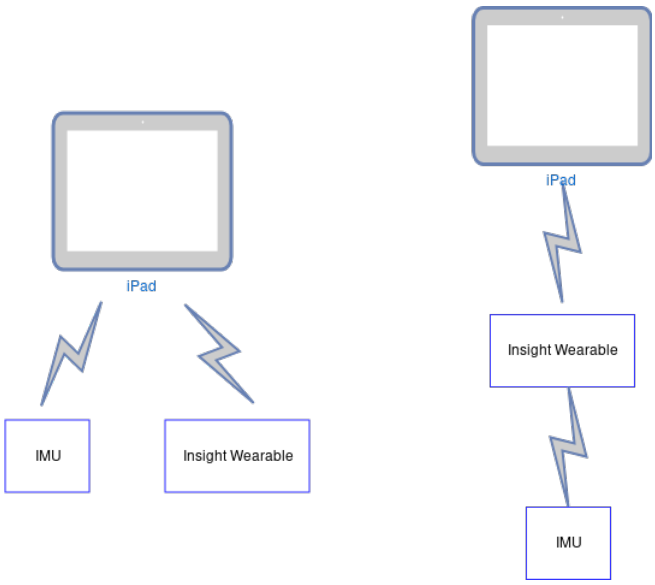

Figure 6.3: Possible Network Topologies

An inconvenient fact regarding a configuration where the *tInterface* is the master of the piconet is that, the master of a piconet is the initiator device, the one which connects to others, unless a role change occurs after connection setup.

Although technically possible, a solution involving this topology produces an array of problems for usability such as, for example, the master (*tInterface*) would have to decide to what iPad it would connect, or have a dynamic or hard coded MAC address in order to facilitate connection initiation. As such, this would create a major problem, effectively making it more viable to use the first alternative, where both devices connect to the iPad separately.

Another possibility, although equal to the previous to the end user, could be using the *tInterface* as both a peripheral and central at the same time. The simplicity to the user would, in fact, make implementation more complex.

6.2 Summary

No assumptions can be made about how the iOS Bluetooth stack handles two different peripherals. However calculations for a favorable case were performed, proving that in theory it is possible to substitute the cable needed for the second IMU by a Bluetooth Low Energy connection and thus creating another, separate device. Also, various network configurations were considered, however using both devices as peripherals seems the most adequate solution in terms of user experience.

Chapter 7

Final Conclusions

As initially described, the BM78 module does not satisfy requirements, due to the number of messages that are not reliably transmitted. The lack of control and openness of this module led to a profound research regarding Bluetooth technology, its market and embedded solutions. A number of tools were also used, some created, in order to be able to test and analyze all of the connections between modules, iPads and PCs. The search for solutions, at first, led to using the WT12 Bluetooth v2.1+EDR module to perform tests in order to evaluate its reliability and throughput capabilities. At first, it proved to be much more reliable and capable than the original BM78 when connected to a PC. Besides the good results, the documentation regarding the necessary coprocessor in order to connect to an iPad indicated that the requirements would never be achieved with the WT12 (or any other BR/EDR module). With this constraint, BLE was studied in order to evaluate its adaptation to the *tInterface*.

After researching and performing tests with different Bluetooth *SoCs* and modules as well as WiFi, results have shown that the establish requirements are possible to achieve and surpass. With the two technologies tested, Bluetooth and Wifi, a big discrepancy can be seen in both throughput, power consumption and usability for both solutions.

Using Bluetooth in order to support Adapttech's product wireless communications is definitely possible and, although throughput values are much lower, it still poses a great advantage in terms of usage of the product. If it was not an absolute requirement for Adapttech using an Apple tablet to communicate with the *tInterface*, the challenge would possibly be less demanding. The lack of documentation, lack of openness and support as well as the restrictive APIs provided and required external hardware, make it very difficult, in Apple environment, to produce quality analysis and improvements to this system with the resources available to a startup company. If using other devices besides the iPad, such as a regular PC or Android tablet, the possibilities for this product would be more broad, as it is far more easy to find documentation with other hardware, apart from Apple. Still, Apple shows improvements in successive models of the its iPad, as Bluetooth throughput and additional features are regularly discussed at developers events. Regarding Bluetooth Classic, as it is much more difficult to analyze, and implement products to work with iPads or iPhones, the natural step is to focus more on Bluetooth Low Energy, as there are no imposed

restrictions. Although, as demonstrated, BR/EDR allows for bigger bandwidths when used with other Bluetooth hardware, specific restrictions taken into account during this dissertation's work, make this technology less advantageous when compared to other.

As with Bluetooth Low Energy, there is no required external hardware or enrollment in specific programs in order to communicate with Apple hardware, there is more support by the engineering community, enabling more developers to develop devices that interact with iOS. Furthermore, with newer Bluetooth specifications, maximum bandwidth has been substantially increased, showing that this is a real interest subject in the Bluetooth community. With the introduction of Bluetooth 5, another step was taken into allowing an ever increasing throughput for BLE devices, making this an attractive and versatile technology, adequate for usage both in very low power devices that can operate for years on a battery, as well with devices with higher throughput requirements.

WiFi, the other wireless communication technology tested, is much more adequate for high performance devices, enabling for a much higher transmission rate than its tested counterparts. WiFi fulfills satisfactory throughput requirements, however, it does not allow for ease of usage for the product on users perspective, requiring an AP to be configured previously and proving more power consuming than the other technologies tested.

A good amount of work was dedicated to circumventing restrictions posed by Apple's hardware, in order to correctly measure throughput and capture/analyze, that otherwise could have been much more easily achieved. The lack of reliable sources of information regarding protocols, the Bluetooth stack used in iOS and the vague official documentation were a constant obstacle throughout this dissertation. A number of different programming languages were used in order to be able to perform the required tests.

Regarding the different *SoCs* and Bluetooth modules tested, not all required the same amount of expertise and experience in order to reach the same goal. The BM78 and WT12 modules are used as *black boxes*, as there is none or little information about how they work and what are their maximum supported data rates or, why they often suffer from a complete system halt during operation. Implementing a solution for this problem with the ESP32 and CC2640R2 allows for a much more detailed perspective of what is really happening, as the two *SoCs* offer debug support and programming is done by the end developer. Although it imposes a harder learning curve, ultimately it allows for much more precise control of the operations carried out, enabling them to be used in more specific applications, in contrary to other more generic Bluetooth modules that only offer a transparent UART service.

A great aid in understanding Bluetooth communications was the Ubertooth, which was able to provide meaningful information when performing tests with BLE devices. Although far from perfect, this device is fairly able to support most applications it was designed for. However, for more detailed analysis in more demanding cases, such as developing and certifying Bluetooth modules, professional tools are required.

Directly related with this dissertation's work, two prototypes have already been produced by Adapttech, in order to mitigate the throughput problems existent in the current system. Both the

ESP32 and a CC2640R2 based module have been utilized in each of the prototypes. After successfully testing and fulfilling requirements, the next step will be to produce a new version of the *tInterface* using a Sable-X-R2 module, based of TI's CC2640R2. As theoretically proven possible, future work regarding Adapttech's product would be to implement and test the possibility of having an, external, wireless IMU. Also, a detailed understanding of the inner works of Bluetooth technology was acquired in order to allow enhancement of specific features of its Adapttech's product.

Ultimately, having achieved requirements and, with a perspective of a favorable evolution, the most appropriate wireless technology found for this product specific needs was Bluetooth Low Energy. Additionally, a specific Bluetooth *SoC* was found to have the appropriate characteristics in order to be integrated into the *tInterface* system, providing a very low power and well supported platform, already supporting the latest version of the Bluetooth specification.

Appendix A

A.1 Python code used in tests with PC.

```
\lstset{breaklines=true}
\begin{lstlisting}[language=Python]
import os
import sys
import bluetooth
import time
import struct
import select
import signal
import time
from chronometer import Chronometer

startrealtime = bytes.fromhex('42470A0000004E440A')
ack = bytes.fromhex('42470A0400004E440A')
stop = bytes.fromhex('42470A0300004E440A')
target='all'
last_msg_time=0
msg_time = 0
error_counter=0
error_counter1=0
biggest_gap=0
total_gaps=0
bytes_recvd=0
freq=10
i=0
j=0
start_time=0
elapsed_time=0
```

```
s = bluetooth.BluetoothSocket( bluetooth.RFCOMM)

def signal_handler(signal, frame):
os.system('clear')
print("________Test Report________\n")
print("Test Time: ",int(test_time/60),":",int(test_time%60))
print("Total Messages Received: ",i)
print("TimeStamps Skipped = ",error_counter)#,"(",round(error_counter/i,3),")%")
print("Message Errors = ",j-i)#,"(",round(j-i/i,3),")%")
print("Biggest time gap (ms) = ",biggest_gap)
print("Total time gaps (ms) = ",total_gaps)#,"(",round(total_gaps/test_time,3),")%")
print("Total Bytes Received: ",bytes_recvd)
print("Average Troughput: ",round(8*(bytes_recvd/(test_time)/1000),2),"Kb/s")
print('Sending Stop Message')
print('Connection Closed!')
s.send(stop)
s.close()
sys.exit(0)


def recvall(sock, n):
    # Helper function to recv n bytes or return None if EOF is hit
    data = b''
    while len(data) < n:
        packet = sock.recv(n - len(data))
        if not packet:
            return None
        data += packet
    return data

def search_header(sock, n):
    # Helper function to recv n bytes or return None if EOF is hit
    data = b''
    header_ok=0
    while header_ok == 0:
        packet = recvall(1)
        #if packet == 66:

        if not packet:
            return None
```

```
        data += packet
    return data

def read_msg(sock,n,previousT):




msg_pdu=recvall(s,n+3)
timestamp_tuple=struct.unpack('I',msg_pdu[0:4])
TimeStamp=int(timestamp_tuple[0])


print("Header Correct \n")
print("Length: ",n+4)
print("Message Number: ",i)
print("TimeStamp = ",TimeStamp)
print("TimeStamps Skipped = ",error_counter)
print("Message Errors = ",j-i)
print("Biggest time gap (ms) = ",biggest_gap)
print("Total time gaps (ms) = ",total_gaps)
print("Average Troughput: ",round(8*(bytes_recvd/(test_time)/1000),2),"Kb/s")
print("Header: ",recvbuffer)
print("PDU: ", msg_pdu)
print("")
return TimeStamp

serverMACAddress = '00:07:80:4A:10:ED'




while 1:
print("Choose Options:")
print("1-Start Real Time")
print("2-Scan Devices And Services")
#print(ord('a'))
text = input() # Note change to the old (Python 2) raw_input
if text == "1" :
```

```
port = 1


s.connect((serverMACAddress, port))
signal.signal(signal.SIGINT, signal_handler)
print('Press Ctrl+C to Exit')
#signal.pause()
#s.setblocking(0)
s.send(startrealtime)
time.sleep(2)
s.send(ack)
start_time = time.time()
while(1) :
try:
test_time=time.time() - start_time
#os.system('clear')
print("\n\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nTest
Time: ",int(test_time/60),":",int(test_time%60))
j=j+1
recvbuffer=recvall(s,3)
if recvbuffer[0]==66 and recvbuffer[1]==71 and recvbuffer[2]==10 :
i=i+1
last_msg_time=msg_time
header_msg=recvall(s,3)
bytes_recvd+=9
packet_size_tuple=struct.unpack('h',header_msg[1:3])
packet_size=int(packet_size_tuple[0])
msg_time=read_msg(s,packet_size,msg_time)
bytes_recvd+=packet_size
if(msg_time-last_msg_time>freq and last_msg_time!=0):
error_counter=error_counter+1
total_gaps+=msg_time-last_msg_time
if(msg_time-last_msg_time>biggest_gap):
biggest_gap=msg_time-last_msg_time
elif recvbuffer[1]==66 and recvbuffer[2]==71 :
i=i+1
last_msg_time=msg_time
header_msg=recvall(s,4)
bytes_recvd+=10
```

```
packet_size_tuple=struct.unpack('h',header_msg[2:4])
packet_size=int(packet_size_tuple[0])
msg_time=read_msg(s,packet_size,msg_time)
bytes_recvd+=packet_size
if(msg_time-last_msg_time>freq and last_msg_time!=0):
error_counter=error_counter+1
total_gaps+=msg_time-last_msg_time
if(msg_time-last_msg_time>biggest_gap):
biggest_gap=msg_time-last_msg_time
elif recvbuffer[2]==66 :
i=i+1
last_msg_time=msg_time
header_msg=recvall(s,5)
bytes_recvd+=11
packet_size_tuple=struct.unpack('h',header_msg[3:5])
packet_size=int(packet_size_tuple[0])
msg_time=read_msg(s,packet_size,msg_time)
bytes_recvd+=packet_size
if(msg_time-last_msg_time>freq and last_msg_time!=0):
error_counter=error_counter+1
total_gaps+=msg_time-last_msg_time
if(msg_time-last_msg_time>biggest_gap):
biggest_gap=msg_time-last_msg_time
else:
print("Incorrect Header")
except bluetooth.btcommon.BluetoothError as error:
print ("Caught BluetoothError: ", error)
s.close()
s.connect((serverMACAddress, port))
time.sleep(5)
pass

#print(recvbuffer)


s.send(stop)
s.close()
```

```
if text == "2" :
target = "all"
if target == "all": target = None

services = bluetooth.find_service(address=target)

if len(services) > 0:
    print("found %d services on %s" % (len(services), sys.argv[1]))
    print("")
else:
    print("no services found")

for svc in services:
    print("Service Name: %s"    % svc["name"])
    print("    Host:        %s" % svc["host"])
    print("    Description: %s" % svc["description"])
    print("    Provided By: %s" % svc["provider"])
    print("    Protocol:    %s" % svc["protocol"])
    print("    channel/PSM: %s" % svc["port"])
    print("    svc classes: %s "% svc["service-classes"])
    print("    profiles:    %s "% svc["profiles"])
    print("    service id:  %s "% svc["service-id"])
print("")

#s.send(text)
sock.close()


\end{lstlisting}}
```

References

[1] Intel. Different Wi-Fi Protocols and Data Rates. URL: https://www.intel.com/content/www/us/en/support/articles/000005725/network-and-i-o/wireless-networking.html.

[2] Bluetooth Special Interest Group. Bluetooth Core Specification Version 4.2. *Bluetooth Core Specification Version 4.2*, (December):2684, 2014. URL: https://www.bluetooth.org/en-us/specification/adopted-specifications.

[3] Karl Torvmark. Three flavors of Bluetooth ® : Which one to choose? The current state of Smart. URL: http://www.ti.com/lit/wp/swry007/swry007.pdf.

[4] Serial Port Emulation. RFCOMM WITH TS 07 . 10 Abstract. 2012.

[5] Keith Palmgren. Diffie-Hellman Key Exchange – A Non-Mathematician's Explanation. *ISSA Journal*, pages 1–7, 2006. URL: http://academic.regis.edu/cias/ia/palmgren{_}-{_}diffie-hellman{_}key{_}exchange.pdf.

[6] John Padgette, John Bahr, Mayank Batra, Marcel Holtmann, Rhonda Smithbey, Lily Chen, and Karen Scarfone. NIST Special Publication 800-121 Revision 2 Guide to Bluetooth Security. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-121r2.pdf, doi:10.6028/NIST.SP.800-121r2.

[7] Diagnostic Horizon and Scanning Centre. Diagnostic Technology: iPhone, iPod and iPad add-on or plug-in medical devices Clinical Question:. (December), 2012.

[8] S. Stowe and S. Harding. Telecare, telehealth and telemedicine. *European Geriatric Medicine*, 1(3):193–197, 2010. URL: http://dx.doi.org/10.1016/j.eurger.2010.04.002, doi:10.1016/j.eurger.2010.04.002.

[9] Rim Negra, Imen Jemili, and Abdelfettah Belghith. Wireless Body Area Networks: Applications and Technologies. *Procedia Computer Science*, 83:1274–1281, 2016. URL: http://dx.doi.org/10.1016/j.procs.2016.04.266, doi:10.1016/j.procs.2016.04.266.

[10] Junn Yen Hu and Chun Chuan Yang. On the design of mobility management scheme for 802.16-based network environment. *IEEE Vehicular Technology Conference*, 2:720–724, 2015. doi:10.1109/VETECF.2005.1558018.

[11] Humaira Abdus Salam and Bilal Muhammad Khan. Use of wireless system in healthcare for developing countries. *Digital Communications and Networks*, 2(1):35–46, 2016. URL: http://dx.doi.org/10.1016/j.dcan.2015.11.001, doi:10.1016/j.dcan.2015.11.001.

[12] Gaurav Bora, Saurabh Bora, Shivendra Singh, and Sheikh Mohamad Arsalan. OSI Reference Model Networking : An Overview. *International Journal of Computer Trends and Technology*, 7(4):214–218, 2014.

[13] Dai Davis. Bluetooth. *Network Security*, 2002(4):11–12, apr 2002. URL: https://www.sciencedirect.com/science/article/pii/S1353485802004130, doi:10.1016/S1353-4858(02)00413-0.

[14] Bluetooth Special Interest Group. Why Build with Bluetooth | Bluetooth Technology Website. URL: https://www.bluetooth.com/develop-with-bluetooth/why-build-with-bluetooth.

[15] ANT+. This is ANT, 2017. URL: http://www.thisisant.com/.

[16] CISCO. Wireless Technologies. *Cisco*, pages 1–41, 2005. URL: http://www.igi-global.com/chapter/advances-security-privacy-wireless-sensor/58886, doi:10.4018/978-1-61350-101-6.

[17] Medium Access Control, Application Support, and Access Control Lists. Key Standards and Industry Specifications. pages 7–15, 2012.

[18] Boris Bellalta, Luciano Bononi, Raffaele Bruno, and Andreas Kassler. Next generation IEEE 802.11 Wireless Local Area Networks: Current status, future directions and open challenges. *Computer Communications*, 75:1–25, 2016. URL: http://dx.doi.org/10.1016/j.comcom.2015.10.007, doi:10.1016/j.comcom.2015.10.007.

[19] Intel. Ultra-Wideband (UWB) Technology White Paper.

[20] DecaWave. DecaWave | Next Frontier of Wireless Technology, 2016. URL: https://decawave.com/technology.

[21] Bluegiga Technologies Ltd. How to speed up iAP data stream. URL: https://www.silabs.com/community/wireless/bluetooth/forum.topic.html/how{_}to{_}speed{_}up{_}iap-gGYq.

[22] Apple Inc. Bluetooth Accessory Design Guidelines for Apple Products. *History*, page 24, 2011.

[23] Ubertooth. GitHub - greatscottgadgets/ubertooth: Software, firmware and hardware designs for Ubertooth. URL: https://github.com/greatscottgadgets/ubertooth.

[24] Oracle General Ledger. Data Sheet. *Workbench*, 0402(April):1–8, 2014. doi:10.1007/978-3-211-89836-9_355.

[25] Medicines and Healthcare Products Regulatory Agency. Guidance on the regulations for electronic instructions for use of medical devices. (February), 2015.

[26] Riku MettŠlä, Olof Dellien, and Johan Sšrensen. Serial Port Profile. 2012.

[27] Texas Instruments. CC2540 and CC2541 Bluetooth ® low energy Software Developer's Reference Guide. 2010. URL: http://www.ti.com/lit/ug/swru271g/swru271g.pdf.

[28] Texas Instruments. CC256x MSP430 TI's Bluetooth Stack Basic SPPDemo APP Improving throughput v14 - Texas Instruments Wiki. URL: http://processors.wiki.ti.com/index.php/CC256x{_}MSP430{_}TI{%}27s{_}Bluetooth{_}Stack{_}Basic{_}SPPDemo{_}APP{_}Imp

[29] Bluetooth Special Interest Group. Exploring Bluetooth 5 - Going the Distance | Bluetooth Technology Website. URL: http://blog.bluetooth.com/exploring-bluetooth-5-how-fast-can-it-behttps://blog.bluetooth.com/exploring-bluetooth-5-going-the-distance.

[30] J.A.a Afonso, A.J.F.a Maio, and R.b c Simoes. Performance Evaluation of Bluetooth Low Energy for High Data Rate Body Area Networks. *Wireless Personal Communications*, 90(1):121–141, 2016. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84964286066{&}partnerID=40{&}md5=9fa7a0844709a9118ee548b5050e0845, doi:10.1007/s11277-016-3335-4.

[31] Apple Inc. What ' s New in Core Bluetooth Best practices. 2017. URL: https://devstreaming-cdn.apple.com/videos/wwdc/2017/712jqzhsxoww3zn/712/712{_}whats{_}new{_}in{_}core{_}bluetooth.pdf?dl=1.

[32] Tolga Girici, Chenxi Zhu, Jonathan R Agre, and Anthony Ephremides. Proportional Fair Scheduling Algorithm in OFDMA-Based Wireless Systems with QoS Constraints. *JOURNAL OF COMMUNICATIONS AND NETWORKS*, 12(1), 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.6859{&}rep=rep1{&}type=pdf.