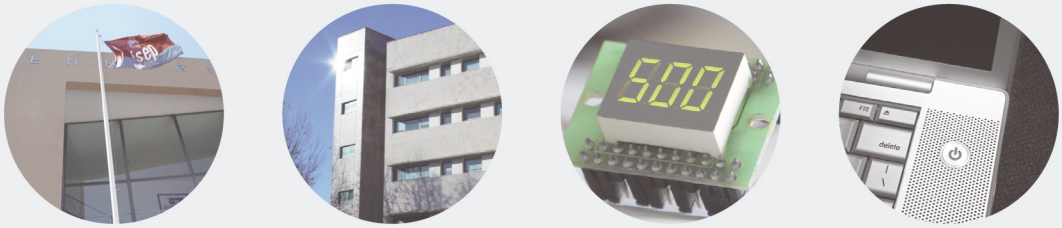
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Sistema de Comunicação baseado na Tecnologia LoRa para Aplicações IoT

FLAVIO PEREIRA BRANCO
novembro de 2020

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES




Sistema de Comunicação baseado na Tecnologia LoRa para Aplicações IoT

FLAVIO PEREIRA BRANCO
Outubro de 2020

ENGENHARIA DO PORTO
Aplicações IoT
Outubro de 2020

POLITÉCNICO
DO PORTO

Sistema de Comunicação baseado na Tecnologia LoRa para Aplicações IoT

Flávio Pereira Branco

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Flávio Pereira Branco, Nº 1150880, 1150880@isep.ipp.pt
Orientação científica: Lino Manuel Baptista Figueiredo, lbf@isep.ipp.pt
Co-orientação científica: António José Matos De Meireles, aem@isep.ipp.pt
Co-orientação científica: António Constantino Lopes Martins, acm@isep.ipp.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
11 de novembro de 2020

Agradecimentos

Com a realização e conclusão desta dissertação termina simultâneamente mais um capítulo da minha vida académica. Um percurso difícil porém aligeirado pelo apoio e incentivo de diversas pessoas. Como tal, aproveito aqui para manifestar o meu maior agradecimento a algumas dessas pessoas que contribuíram para este sucesso.

Em primeiro lugar, gostaria de expressar o meu agradecimento ao Dr. Lino Figueiredo e ao Engenheiro António Meireles pela oportunidade de realizar este trabalho, pela disponibilidade, orientação, contributo e boa disposição que demonstraram ao longo do trabalho e que permitiu que este decorre-se da melhor forma possível.

De seguida, gostaria de agradecer aos meus amigos, pelas opiniões, incentivos e bons momentos que permitiram tornar esta etapa académica especial e ainda mais agradável.

Por último, gostaria de oferecer o agradecimento mais importante aos meus pais e irmão, por todo o esforço, preocupação, incentivo e inúmeros outros aspetos fundamentais que me permitiram atingir este objetivo, pois só eles sabem puramente o que foi necessário para chegar a este ponto.

Os meus sinceros agradecimentos!

Resumo

Outrora, a ideia de interconectar "todas" as coisas não passava de um conceito porém, nos últimos anos, esta ideia tem-se tornado uma realidade estando cada vez mais presente no nosso quotidiano. Ano após ano o número de dispositivos conectados tem vindo a aumentar significativamente obrigando a tecnologia a evoluir e a adaptar-se a esta tendência.

O conceito da Internet das Coisas afetou diversas áreas e prevê-se que, dentro destas, a área dos cuidados médicos e de saúde seja uma com grande potencial para se desenvolver e que terá grande impacto no futuro.

Este aumento no número de dispositivos conectados e as cararísticas favoráveis que as comunicações sem fios apresentam resultaram numa dominância no mercado e numa das grandes apostas por parte das empresas no que diz respeito à forma como equipamentos comunicam entre si. No entanto, com a enorme diversidade de aplicações, torna-se difícil a escolha de uma tecnologia que se adeque a todas estas. Atualmente, o mercado dispõe de diversas tecnologias com diferentes caraterísticas que melhor se adequam a determinadas soluções e ambientes.

Neste trabalho é estudada a tecnologia de comunicação sem fios Long Range para posteriormente ser aplicada a um sistema *healthcare* simples baseado na filosofia da Internet das Coisas. Apesar de o sistema *healthcare* ser simples, o conceito da Internet das Coisas possui um certo nível de complexidade devido ao número de componentes e, consequentemente, às respetivas tecnologias associadas a estes. Ao longo deste documento é descrito o processo de desenvolvimento deste sistema. Será abordada a integração da tecnologia Long Range, assim como um outro conjunto de tecnologias inerentes ao conceito da Internet das Coisas que permitiram desenvolver um sistema *healthcare* simples baseado na filosofia da Internet das Coisas.

Palavras-Chave: Internet da Coisas (IoT), Comunicações sem fios, Sistemas *Healthcare*, Long Range (LoRa), Baixo consumo de energia.

Abstract

In the past, the idea of interconnecting "all" things was just a concept, but in recent years, this idea has become a reality and is increasingly present in our daily lives. Year after year the number of connected devices has been increasing significantly forcing technology to evolve and adapt to this trend.

The concept of the Internet of Things has affected several areas and it is expected that the area related to healthcare to be one with great potential for development and that will have a great impact in the future.

This increase in the number of connected devices and the favorable characteristics that wireless communications have resulted in market dominance and one of the big bets on the part of companies with regard to the way equipment communicates with each other. However, with the enormous diversity of applications, it is difficult to choose a technology that fits all of these. Currently, the market offers several technologies with different characteristics that are best suited to certain solutions and environments.

In this work, the Long Range wireless communication technology is studied to later be applied to a simple healthcare system based on IoT philosophy. Although the healthcare system is simple, the concept of the Internet of Things has a certain level of complexity due to the number of components and, consequently, to the respective technologies associated with them. Throughout this document the process of developing this system is described. The integration of the Long Range technology will be addressed, as well as another set of technologies inherent to the concept of the Internet of Things that allowed the development of a simple healthcare system based on the Internet of Things philosophy.

Keywords: Internet of Things (IoT), Wireless communication, Healthcare systems, Long Range (LoRa), Low power consumption.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

Abreviatura	Descrição
ADC	*Analog to Digital Converter*
AES	*Advanced Encryption Standard*
AMQP	*Advanced Message Queueing Protocol*
AP	*Access Point*
BLE	*Bluetooth Low-Energy*
BW	*Bandwidth*
CF	*Carrier Frequency*
CoAP	*Constrained Application Protocol*
CR	*Coding Rate*
CRC	*Cyclic Redundancy Check*
CSS	*Chirp Spread Spectrum*
DTLS	*Datagram Transport Layer Security*
ECG	*Eletrocardiografia*
EEG	*Eletroencefalografia*
EMG	*Eletromiografia*
EU	*Europe Union*
EUI	*Extended Unique Identifier*
FDM	*Frequency Division Multiplexing*
FIFO	*First In First Out*
FSK	*Frequency Shift Keying*
GSM	*Global System for Mobile Communications*
GSR	*Galvanic Skin Response*
HTTP	*Hypertext Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IIoT	*Industrial Internet of Things*
IoE	*Internet of Everything*
IoT	*Internet of Things*
ISM	*Industrial, Scientific and Medical*
ITU	*International Telecommunication Union*
LNA	*Low Noise Amplifier*
LoRa	*Long Range*

Abreviatura	Descrição
LPWAN	*Low Power Wide Area Network*
LTE	*Long Term Evolution*
MIC	*Message Integrity Code*
MQTT	*Message Queue Telemetry Transport*
M2M	*Machine to Machine*
NATO	*North Atlantic Treaty Organization*
NFC	*Near Field Communication*
P2P	*Peer to Peer*
PKS	*Phase Shift Keying*
PLC	*Programmable Logic Controller*
QoS	*Quality of Service*
QPSK	*Quadrature Phase-Shift Keying*
RFID	*Radio FrequencyIdentification*
SF	*Spreading Factor*
SPI	*Serial Peripheral Interface*
SoC	*System on a Chip*
TCP	*Transmission Control Protocol*
TDM	*Time Division Multiplexed/Multiplexing*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
UI	*User Interface*
UMTS	*Universal Mo-bile Telecommunications System*
URI	*Uniform Resource Identifier*
US	*United States*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*
WBAN	*Wireless Body Area Netowrks*
WDA	*Wireless Device Access*
Wi-Fi	*Wireless-Fidelity*
WLAN	*Wireless Local Area Network*
WPAN	*Wireless Personal Area Network*
WSN	*Wireless Sensor Network*
WWAN	*Wireless Wide Area Network*

Capítulo 1

Introdução

O presente documento trata o trabalho realizado no âmbito da unidade curricular Tese/Dissertação do 2º ano do Mestrado em Engenharia Eletrotécnica e de Computadores - Automação e Sistemas.

Neste capítulo é realizada uma contextualização relativa ao tema do trabalho, assim como os aspetos que motivaram o seu desenvolvimento. De seguida, são referidos os objetivos pretendidos para este, bem como uma calendarização que permitiu organizar e planear a execução de todas as tarefas. Por último, é realizada uma breve descrição acerca dos diferentes capítulos, de forma a facilitar a compreensão de cada um destes e da estrutura do documento.

1.1 Contexto

O número de dispositivos conectados à Internet das Coisas (IoT) tem vindo a aumentar ao longo dos anos e prevê-se que esta tendência se venha a manter no futuro. No final de 2020 é esperado que existam mais de 26 mil milhões de dispositivos conectados [1]. Este número terá grande impacto não só nas empresas mas também no dia a dia das pessoas. A conectividade entre os dispositivos é fundamental para a aplicação da IoT porém, devido a este grande número, é previsível que grande parte destes tenha por base uma comunicação sem fios. A rápida evolução deste número levou ao desenvolvimento de novas tecnologias ou a adaptação de outras já existentes.

Os cuidados médicos e de saúde representam um dos segmentos da IoT e prevê-se que este será uma das áreas mais beneficiadas pelo seu desenvolvimento. Prevê-se que em 2025 esta se torne na área de maior impacto económico de entre todos os segmentos, possuindo uma cota no mercado de 41 % [2].

Atualmente, apesar dos avanços tecnológicos ocorridos nos últimos anos, ainda não existe uma tecnologia de comunicação sem fios que se adeque idealmente a todo o tipo de soluções, no entanto existem diversas opções disponíveis no mercado com determinadas carateristicas que as tornam bastante atrativas para certas aplicações. Neste trabalho pretende-se efetuar o estudo de uma destas tecnologias, nomendamente o Long Range (LoRa), e efetuar a sua integração com a área dos cuidados médicos e de saúde da IoT.

1.2 Objetivos

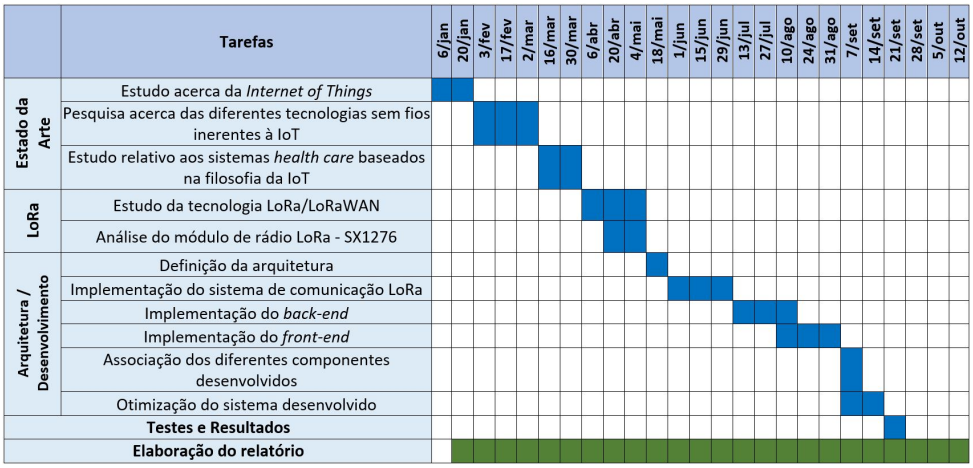
O objetivo principal deste trabalho incide sobre o estudo acerca da tecnologia LoRa e integrá-la num sistema *healthcare* simples baseado na filosofia da IoT. Este objetivo pode ser dividido nos seguintes subobjetivos de forma a facilitar a estruturação do trabalho:

- Pesquisa acerca da IoT e dos diferentes conceitos inerentes a este.
- Estudo das diferentes tecnologias sem fios existentes no mercado que possibilitam a aplicação do conceito da IoT.
- Pesquisa acerca dos sistemas *healthcare* baseados na filosofia da IoT.
- Estudo acerca da tecnologia LoRa.
- Definição da arquitetura do sistema.
- Implementação dos transceptores LoRa.
- Desenvolvimento dos diferentes sistemas de comunicação.
- Desenvolvimento do *front-end* para exposição dos dados.

1.3 Calendarização

Na Tabela 1.1 encontra-se representado a divisão das diferentes tarefas envolvidas na elaboração deste projeto, bem como a sua calendarização.

Tabela 1.1: Calendarização das tarefas.



1.4 Organização da Dissertação

No primeiro capítulo é realizada uma introdução ao projeto desenvolvido. Neste é apresentada uma pequena contextualização, os objetivos, a calendarização e a estrutura do trabalho.

No capítulo seguinte, denominado de Estado da Arte, é realizado um estudo acerca do conceito da Internet das Coisas e de alguns conceitos relacionados com as comunicações sem fios. Ainda neste capítulo é realizada uma pesquisa acerca das diferentes tecnologias de comunicação sem fios existentes atualmente para a aplicação da Internet das Coisas, bem como de um dos seus segmentos nomeadamente os sistemas *healthcare*.

O terceiro capítulo, designado de LoRa/LoRaWAN, diz respeito ao estudo efetuado acerca da tecnologia LoRa/LoRaWAN. Neste é também realizada a análise ao *datasheet* de um módulo de rádio LoRa de modo a entender o seu funcionamento.

O capítulo quatro, Arquitetura do Sistema, tem como objetivo abordar a arquitetura da solução, assim como os diversos conceitos e tecnologias envolvidos.

No quinto capítulo, Implementação do Sistema, é exposto todo o processo de implementação. Este inclui o *hardware* e *firmware* desenvolvido para ambos os transceptores, assim como o *software* implementado ao nível do *back-end* e *front-end*.

Com o capítulo seis, denominado de Testes e Resultados, pretende-se demonstrar os resultados obtidos com a solução implementada.

Por último, no capitulo 7 são tiradas as conclusões acerca do trabalho desenvolvido. Além disso são ainda referidas as principais dificuldades sentidas bem como possíveis melhorias em relação ao trabalho realizado.

Capítulo 2

Estado da arte

Atualmente a internet atingiu grande parte do globo afetando a vida humana de diversas maneiras. Encontrámo-nos numa época em que a conectividade é cada vez mais importante e o número de dispositivos conectados à internet é cada vez maior. Este panorama levanta o tema da Internet das Coisas (IoT), que pode ser designado, de uma forma geral, como uma rede de objetos físicos conectado por uma rede (internet ou rede privada). Este tema não é recente, uma vez que os dispositivos comunicam entre si já há alguns anos, no entanto voltou a ser um tema bastante discutido devido às vantagens que a IoT pode trazer não só ao nível das empresas, mas também ao dia a dia das pessoas [3].

2.1 Internet das Coisas

Existem diversas organizações que trabalham sobre o conceito da IoT e que propuseram soluções para este tema. A abordagem neste trabalho será estabelecida segundo a definição da IEEE que define a Internet da Coisas como uma rede de sensores, atuadores e objetos inteligentes cujo objetivo é interconectar "todas" as coisas, incluindo não só os objetos do quotidiano mas também os industriais, de maneira a torná-los inteligentes, programáveis e com uma maior capacidade de interagirem com humanos e entre si [4][5].

Por vezes o conceito da IoT é confundido com outros conceitos que, apesar de possuírem alguns aspetos em comum, o significado final não é o mesmo. Na Figura 2.1 é apresentado um diagrama com diferentes conceitos e o que estes afetam, de modo a eliminar estas ambiguidades.

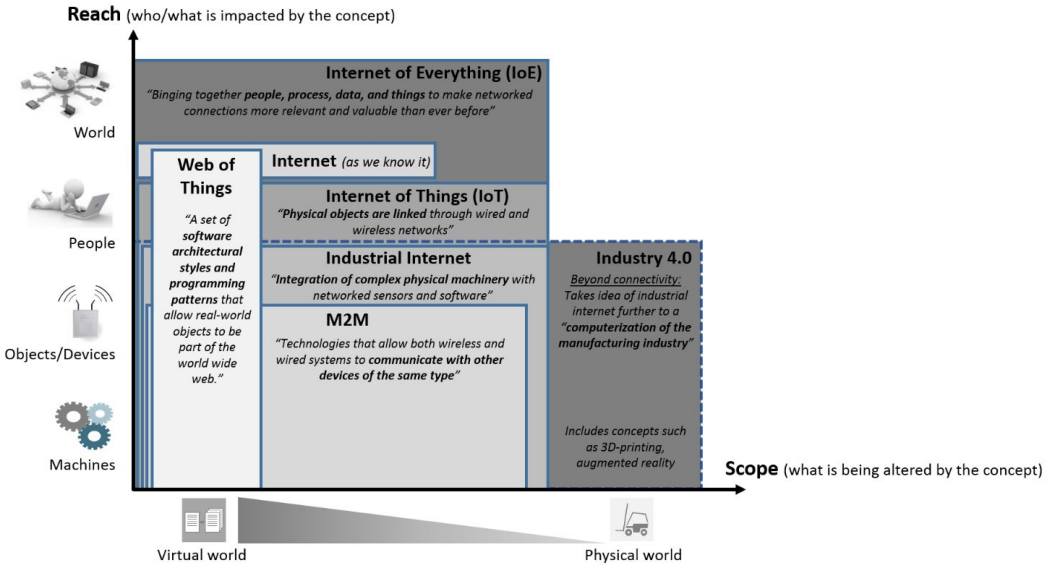


Figura 2.1: Diagrama com diferentes conceitos e o que é afetado por estes [6].

O termo *Machine to Machine* (M2M) é um dos conceitos mais conhecidos, sendo usado à mais de uma década. Este refere-se a tecnologias que permitem, tanto a sistemas com fios como sem fios, comunicarem com outros dispositivos que possuem a mesma capacidade. Inicialmente as comunicações M2M eram apenas utilizadas para conectar duas máquinas, no entanto, com evolução da tecnologia, foi possível a comunicação entre múltiplas máquinas e a distâncias cada vez maiores. O conceito *Industrial Internet*, também denominado de *Industrial Internet of Things* (IIoT), vai além do M2M onde não só as máquinas se encontram interligadas, mas também outros dispositivos tais como sensores e computadores. Estes associados a softwares de gestão permitem recolher mais facilmente informação que pode ser utilizada para otimizar a produtividade e a eficiência de determinados processos provocando, consequentemente, benefícios económicos. A *Industry 4.0*, também designada de quarta revolução industrial, engloba os dois conceitos anteriores, no entanto esta tem como objetivo permitir que o próprio sistema, constituído por sensores, máquinas, computadores, entre outros dispositivos, tomem decisões sem a necessidade de envolvimento humano. Estas decisões serão possíveis através da grande quantidade de informação que estes sistemas conseguem gerar, combinados com técnicas de *machine learning*. Com o aumento da quantidade de informação que estes sistemas recolhem, estes tornar-se-ão cada vez mais inteligentes, provocando cada vez melhores decisões e resultando num aumento da eficiência e da produtividade da fábrica, sendo estes um dos objetivos principais da *Industry 4.0* [6].

Em comparação aos conceitos já abordados, a IoT possui um âmbito superior indo para além do ambiente industrial, atingindo as pessoas através de objetos do quotidiano. A partir da Figura 2.1 é possível observar que o conceito da *Internet*, como todos nós a conhecemos, abrange uma pequena área, uma vez que o seu objetivo é apenas interligar pessoas. Assim como a *Internet*, a *Web of Things* é outro conceito com um âmbito pequeno comparativamente com os outros. De uma forma genérica, a *Web of Things* preocupa-se essencialmente com a arquitetura de *software*. Por último, o conceito de *Internet of Everything* (IoE), apesar de ainda ser muito vago, visa ir além da IoT incluindo todo o tipo de conexões existentes atualmente e no futuro. Nos dias de hoje, a IoE não é nada mais do que uma filosofia de uma tecnologia futura, no entanto será resultado de uma tecnologia do presente, a IoT, e que esta, com o passar do tempo, terá cada vez mais influência no nosso quotidiano.

A IoT define as "coisas" como uma entidade ou um objeto físico que possui uma identificação única, um sistema embebido com a capacidade de transferir dados através da rede. Pode também possuir um atuador de modo a permitir que uma coisa realize uma ação, como por exemplo, ligar ou desligar luzes, aumentar e diminuir a velocidade de rotação de um motor, entre outras. Este conceito de coisa engloba todo o tipo de objetos, desde os mais simples aos mais complexos, como por exemplo, *smartphones*, eletrodomésticos, veículos, iluminação pública, equipamento médico, máquinas de produção, edifícios e muito mais. Além disto, os sensores não necessitam de estar fisicamente ligados ao objeto, por vezes apenas é necessário monitorizar o ambiente em redor da coisa [3].

Segundo [1], prevê-se que haverá mais de 26 mil milhões de dispositivos conectados até ao final de 2020. Este grande número de dispositivos está mudando o panorama tanto para as empresas como para os consumidores possibilitando não só conectar pessoas e sistemas, mas também recolher informação que poderá ser analisada e partilhada. Atualmente já existem várias centenas de projetos IoT anunciados publicamente por parte de empresas e estendem-se em diversas áreas. Na Figura 2.2 é possível observar a percentagem de projetos IoT publicamente anunciados em 2018 nas suas diferentes áreas. Estas áreas são normalmente designadas de categorias ou segmentos da IoT [7].

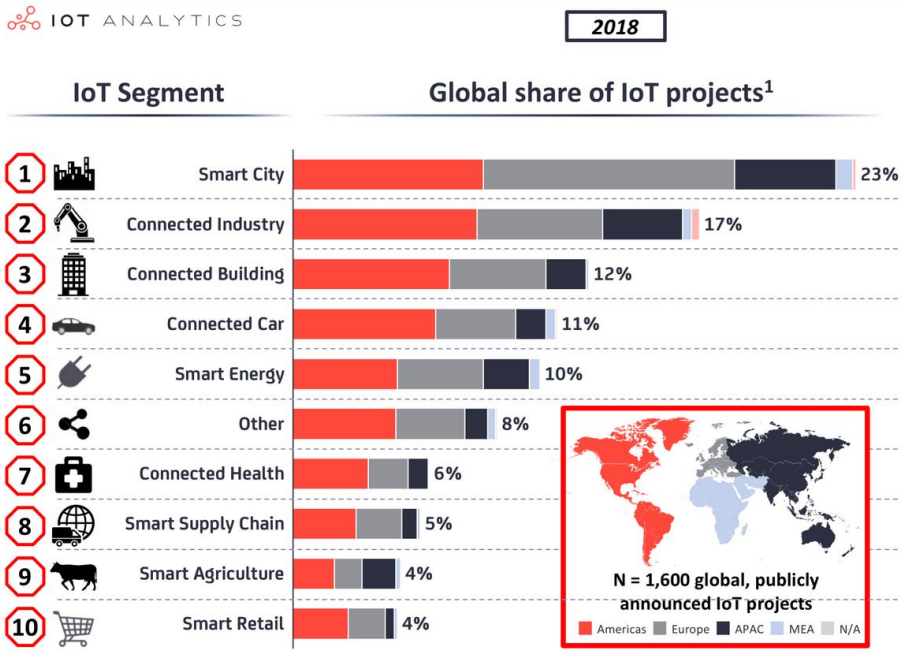


Figura 2.2: Projetos IoT publicamente anunciados em 2018 [8].

Em 2018 o maior segmento de aplicação foram as cidades inteligentes (*Smart Cities*) onde foram implementados diversos projetos sobre a gestão de tráfico, monitorização ambiental, controlo de iluminação pública, segurança, entre outros. Sistemas de estacionamento, controlo e monitorização de trânsito, partilha de bicicletas e vias inteligentes para autocarros são apenas alguns dos projetos mais específicos já desenvolvidos e aplicados nesta área.

A área de conectividade industrial (por vezes também designada por *Industrial Internet of Things* – IIoT), de edifícios e de veículos foram outras grandes áreas de desenvolvimento. Em relação à conectividade industrial as aplicações mais populares incidiram sobretudo sobre a monitorização e controlo remoto de maquinaria, como por exemplo de gruas, empilhadoras, furadoras ou até mesmo a totalidade das máquinas presentes em minas, como é o caso do projeto *Cisco Connected Mining* implementado na empresa de mineração Rio Tinto na Austrália e que, através da monitorização e controlo remoto de toda a mina, permitiu aumentar a eficiência e simultâneamente reduzir os custos de produção assim como melhorar a segurança dos trabalhadores [9]. Diversas aplicações na área de automação e controlo em fábricas foram desenvolvidas, desde a monitorização da produção ao controlo remoto de PLC e controlo de qualidade automático.

No que diz respeito à conectividade de edifícios grande parte dos projetos desenvolvidos foram relativos à segurança e automação, sendo esta última com

o objetivo de reduzir os custos de energia. Dois exemplos deste segmento da IoT é o caso do *Marriot Hotel* na China e do Hotel *Marina Bay Sands* em Singapura que, através do controlo ativo do edíficio, permitiu aumentar a eficiência energética diminuindo o consumo de energia até 30 % [10][11].

No caso da conectividade de veículos os projetos incidiram na sua maioria acerca de soluções para a gestão de frotas e diagnóstico de problemas aos veículos. A FleetNest é uma empresa que disponibiliza soluções para a monitorização e gestão de frotas. Esta permite não nó monitorizar em tempo real a localização dos diferentes veículos da empresa, mas também definir rotas e supervisionar, através de sensores, diferentes parâmetros do próprio veículo, como por exemplo temperatura e humidade das câmaras de refrigeração, consumo de combustível, rotação do motor, velocidade, entre outros [12].

Além destas áreas, a área da saúde, dos transportes, da agricultura inteligente, da revenda de produtos, entre outras, são áreas que já se encontram afetadas pela IoT e com a capacidade de se desenvolverem ainda mais no futuro [8]. Em relação há área da agricultura inteligente existem algumas empresas que disponibilizam soluções nesta área, como é o caso da Telit. Esta fornece não só soluções que permitem monitorizar a qualidade do solo, mas também ferramentas de gestão para os sistemas de irrigação que permitem gerir de uma forma mais sustentável os recursos e simultaneamente otimizar a produção das plantações. Utiliza também previsões meteorológicas para que possa antecipar situações prejudiciais às plantações e caso seja necessário tomar medidas preventivas de forma a diminuir os danos causados a estas [13]. Além disto, a Telti fornece ainda sistemas de monitorização para indústria pecuária. Através de sistemas vestíveis para animais, estes permitem medir diversos parâmetros tais como, ritmo cardíaco, pressão arterial, frequência respiratória, temperatura, entre outros, de modo a monitorizar o estado de saúde dos animais e detetar antecipadamente qualquer tipo de problema relacionado com estes para que se possa efetuar o respetivo tratamento. Adicionalmente permitem rastrear a localização de cada um dos animais, assim como monitorizar os ciclos reprodutivos dos animais de modo promover o processo de reprodução de uma forma mais segura e simultâneamente aumentar a taxa de sucesso [14]. No que diz respeito à área de logística e revenda de produtos a AT&T Business é uma empresa que apresenta soluções para a gestão de recursos. Esta oferece um serviço para o rastreamento de qualquer tipo de produto mesmo para os mais sensíveis, uma vez que este incluí sensores de temperatura, humidade, impacto e intensidade luminosa que monitorizam continuamente e garantem que o produto chega ao seu destino nas melhores condições. Todo o processo de envio pode ser supervisionado através de uma aplicação [15].

Há medida que o cenário da Internet das Coisas se vai definindo, as suas aplicações vão sendo divididas em dois grandes grupos consoante os requerimentos, comerciais e técnicos, que estas priorizam. Estes dois grupos são denominados de aplicações críticas da IoT e aplicações em massa da IoT. O primeiro carateriza-se por necessitar de redes com extrema fiabilidade, baixos níveis de latência e, caso necessário, taxas de transmissão elevadas. Exemplos comuns são a condução autónoma ou aplicações relacionadas com os cuidados de saúde, como é o caso de cirurgias remotas. Por outro lado, as aplicações em massa de IoT não necessitam de ser tão sensíveis à latência nem necessitam de uma taxa de transmissão como as anteriores, isto é, podem funcionar com latências superiores e com taxas de transmissão menores, no entanto requerem um *hardware* de baixo custo, com consumos de energia baixos e com uma grande cobertura de rede. As caraterísticas chave para este tipo de aplicações é a longevidade da bateria, uma boa cobertura, baixo custo e flexibilidade no seu desempenho [16].

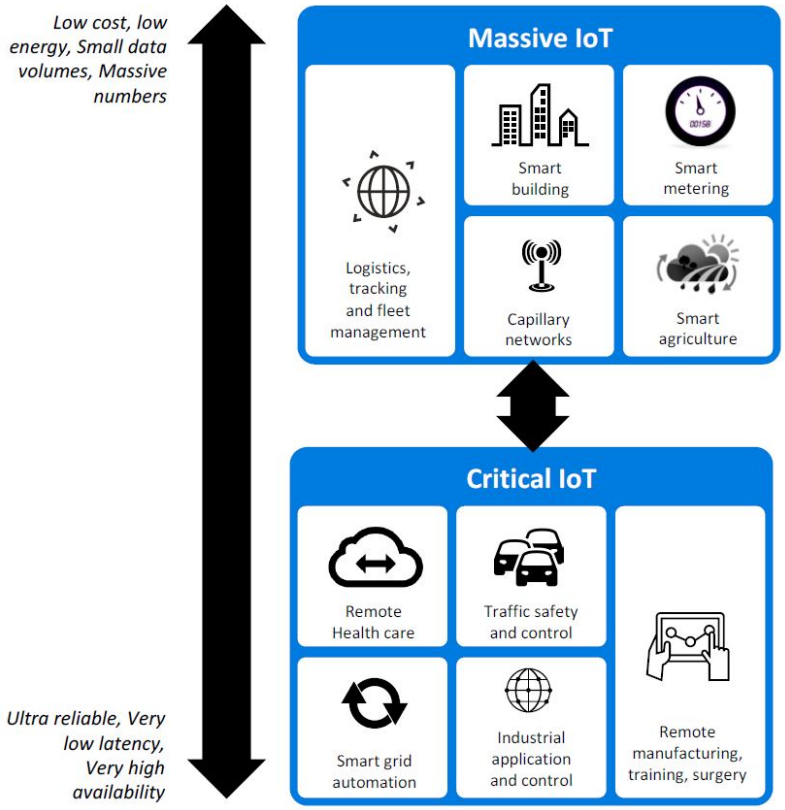


Figura 2.3: Diferentes requerimentos das aplicações em massa e críticas da IoT[17].

2.2 Comunicação Sem Fios

Conectividade entre dispositivos é fundamental para a aplicação da IoT e, como já foi referido anteriormente, nos próximos anos é previsível que o número de dispositivos conectados à IoT aumente rapidamente e que grande parte destes tenha por base uma comunicação sem fios. De uma forma simples a comunicação sem fios, ou *wireless*, refere-se a um método de transmissão de informação entre dois pontos sem a necessidade de utilizar um meio físico, isto é, sem a necessidade de utilizar cabos. Em vez disso, os dados são enviados através de ondas eletromagnéticas (EM). No entanto, ambos os dispositivos necessitam de estar equipados com uma antena, que permite converter sinais elétricos em ondas eletromagnéticas e vice-versa consoante a finalidade do equipamento (emissor, recetor ou transceptor no caso de efetuar ambas as funções).

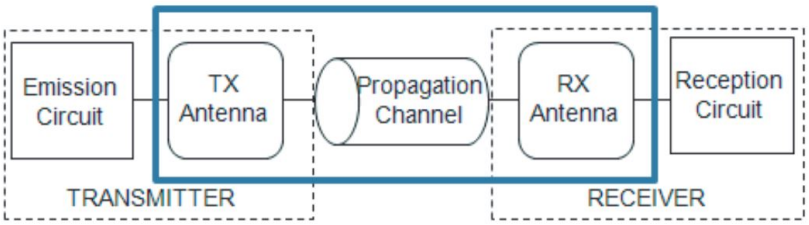


Figura 2.4: Diagrama de blocos de um sistema de comunicação sem fios [18].

2.2.1 Espetro Eletromagnético e Bandas de Rádio

As ondas eletromagnéticas podem ser classificadas segundo a sua energia, frequência ou comprimento de onda. Consoante as suas caraterísticas estas podem ser divididas em 7 grupos, nomeadamente, raios-gama, raio-X, ultravioleta, luz visível, infravermelho, micro-ondas e ondas de rádio. Toda a gama de ondas eletromagnéticas é representada pelo espetro eletromagnético.

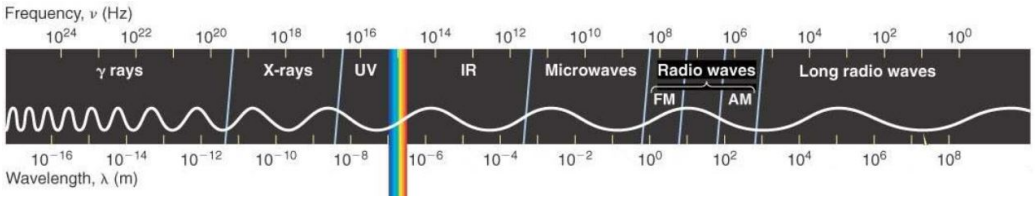


Figura 2.5: Espectro eletromagnético [19].

As ondas de rádio localizam-se num extremo do espetro eletromagnético definido pela gama frequências entre os 3 kHz e o 300 GHz, com o respetivo comprimento de onda entre 100 km e 1 mm. Este tipo de ondas são as mais utilizadas para as comunicações sem fios uma vez que esta gama de frequências é mais facilmente gerada e o grande comprimento de onda que estas possuem permite aumentar a penetrabilidade em obstáculos, aumentando consequentemente a robustez na comunicação. As ondas de rádio podem ainda ser divididas em subgrupos denominadas de bandas de rádio e que constituem o espectro radioelétrico. Em cada banda são tipicamente agrupados serviços semelhantes de modo a prevenir interferências e a aumentar a eficiência de utilização do espectro. Porém, a definição do número de bandas e os respetivos limites não é universal existindo várias entidades, nomeadamente a *International Telecommunication Union* (ITU), IEEE, EU, NATO e US, que organizam o espectro radioelétrico segundo diferentes critérios. A Tabela 2.1 representa a organização do espectro segundo os critérios da ITU.

Tabela 2.1: Designação das bandas de rádio segundo a ITU [20].

Abreviatura	**Descrição**	**Frequência**	**Compri. de onda**
VLF	Very Low Frequency	3-30 kHz	10 -100 km
LF	Low Frequency	30-300 kHz	1-10 km
MF	Medium Frequency	300-3000 kHz	100 -1000 m
HF	High Frequency	3-30 MHz	10 -100 m
VHF	Very High Frequency	30-300 MHz	1-10 m
UHF	Ultra High Frequency	300-3000 MHz	10 – 100 cm
SHF	Super High Frequency	3-30 GHz	1 – 10 cm
EHF	Extremely High Frequency	30-300 GHz	1 – 10 mm

2.2.2 Elementos Básicos de um Sistema de Comunicação Sem Fios

A Figura 2.6 representa o diagrama de blocos de um sistema de comunicação sem fios típico. Este é constituído por três componentes fundamentais existentes em qualquer sistema de comunicação sem fios, um emissor, o canal de transmissão e um recetor. No entanto, é importante salientar que dentro dos diversos sistemas de comunicação sem fios poderão existir particularidades em cada uma das etapas, tanto no processo de transmissão, como de receção.

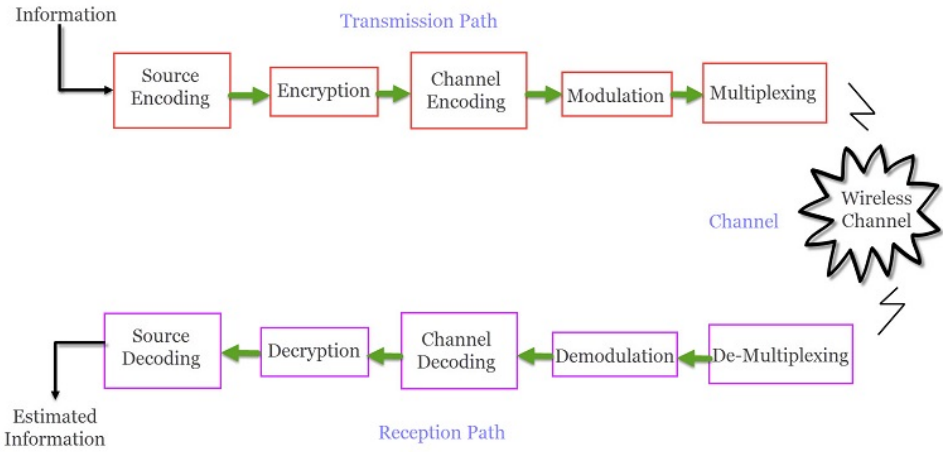


Figura 2.6: Diagrama de blocos de um sistema de comunicação sem fios típico [21].

A processo de transmissão pode ser dividido em cinco etapas. Numa primeira fase, denominada de *Source Encoding*, o sinal original é convertido num outro sinal onde técnicas de processamento de sinal possam ser aplicadas. Um exemplo será a aplicação de um conversor analógico-digital (Analog-to-Digital Converter - ADC) que permitirá converter um sinal analógico num sinal digital para posteriormente possam ser aplicadas outras técnicas tais como amplificação e filtragem. Ainda nesta fase toda a informação redundante do sinal é removida de modo a otimizar o processo de transmissão. De seguida. o sinal passa por uma fase de encriptação onde é utilizado um *standard* de encriptação de forma impedir acessos não autorizados garantindo assim a segurança da informação contida no sinal. Uma vez que, neste tipo de comunicação, a informação não é enviada através de um meio físico, a sua vulnerabilidade a ruídos e a interferências aumenta significativamente. De modo a combater esta debilidade, numa terceira fase é aplicada uma técnica denominada de *Channel Enconding*, que consiste na introdução de uma pequena quantidade de redundância no sinal de forma a torná-lo mais robusto. Após isto, o sinal é modelado através de técnicas de modelação tais como, PKS, FSK, QPSK, entre outras, de modo a facilitar a sua transmissão através de uma antena. Mas antes do envio do sinal e numa última fase, o sinal modelado passa por um processo de multiplexação que, de uma forma sucinta consiste na combinação de múltiplos sinais num só sinal de modo a partilharem a largura de banda, aumentando a eficiência da sua utilização. Nesta fase podem ser utilizadas várias técnicas de multiplexação, nomeadamente *Time Division Multiplexing* (TDM), *Frequency Division Multiplexing* (FDM), entre outras.

O processo de receção, como seria de esperar, consiste no inverso do processo de transmissão. Numa primeira fase é realizada a desmultiplexação, dividindo o sinal proveniente do canal de transmissão nos seus diferentes sinais que o constituem. De seguida é realizado o processo de desmodulação, utilizando técnicas apropriadas, e a redundância introduzida na fase de *Channel Encoding* é removida. Numa quarta fase, o sinal é desencriptado, podendo ser aplicada a última etapa, *Source Decoding*, obtendo-se o sinal original [21].

2.2.3 Vantagens e Desvantagens

A aplicação de um sistemas de comunicação sem fios para a transmissão de informação fornece diversas vantagens, nomeadamente:

- **Mobilidade:** A capacidade mover livremente os dispositivos e estes permanecerem conectados à rede é provavelmente a maior vantagem das comunicações sem fios.
- **Facilidade de instalação:** A configuração e instalação de equipamentos e infraestruturas de uma rede de comunicações sem fios é mais fácil, uma vez que não é necessário instalar cabos. Na generalidade do casos, a configuração de uma rede sem fios é mais fácil do que uma rede com fios.
- **Custo:** Como já foi referido no ponto anterior, este tipo de comunicação sem fios dispensa a utilização de cabos, assim como outros equipamentos associados a estes, diminuindo por sua vez o custo total do sistema. É importante referir que a instalação de uma rede com fios é um processo dispendioso, uma vez que é mais difícil e que consome muito tempo relativamente a uma rede sem fios.
- **Fiabilidade e Capacidade de Recuperação:** Mais uma vez a inexistência de cabos leva a uma maior robustez na rede. A falha da rede por defeito ou avaria de um cabo deixa de ser possível. Além disso a capacidade de recuperação das infraestruturas em caso de fogos, inundações ou outro tipo de desastres é mínimo, devido a um menor número de componentes.

No entanto, assim como todos os sistemas, este também apresenta as suas desvantagens. As desvantagens apresentadas de seguida são algumas das que provocam um maior impacto na escolha e utilização de um sistema de comunicação sem fios:

- **Interferência:** Estes sistemas utilizam um canal de transmissão partilhado e como consequência, existe a probabilidade de um ou mais sinais provenientes de sistemas diferentes provocarem interferência entre si.

- **Segurança:** A segurança é uma das maiores preocupações das comunicações sem fios. Uma vez que a informação é enviada pelo ar, esta pode estar sujeitar a ser intercetada comprometendo a confidencialidade desta.

- **Saúde:** A exposição contínua a qualquer tipo de radiação pode ser prejudicial à saúde. Apesar de os danos causados pelas ondas EM ainda não se encontrarem estabelecidos com precisão, é recomendado que se evite ao máximo.

2.2.4 Modos de Funcionamento e Topologias de uma Rede Sem Fios

Os sistemas de comunicação sem fios podem operar de diversas formas consoante o modo de funcionamento e a topologia escolhida para a rede. Numa rede sem fios existem tipicamente dois componentes essenciais, os clientes (por vezes também designados de nós) e os pontos de acesso (*Access Point* - AP). O cliente é referido como qualquer dispositivo com a capacidade de enviar e receber sinais através de ondas de rádio. Um AP, assim como um cliente, permite enviar e receber sinais, no entanto possui a particularidade de servir como elo de ligação entre os diferentes clientes associados a este AP e a uma rede com fios.

Na generalidade, as comunicações sem fios possuem dois modos de funcionamento, o modo Ad Hoc e Infraestrutura. O modo Ad Hoc é um método que permite estabelecer comunicações sem fios diretas entre clientes. Este tipo de ligações é também por vezes designada de comunicação *Peer-to-Peer* (P2P) e, ao permitir que os clientes operem no modo de funcionamento Ad Hoc, deixa de haver a necessidade de utilizar um AP. Todos os nós de uma rede podem comunicar diretamente entre si, desde que este se encontrem dentro do alcance. É importante referir que este modo de funcionamento é caracterizado por não possuir níveis hierárquicos e a gestão da rede é efetuada por cada um dos dispositivos. Uma rede que opera neste modo de funcionamento é tipicamente utilizada para interligar um baixo número de dispositivos, uma vez que a performance da rede diminuiu com o aumento de nós associados a esta. Ao contrário do que acontece no modo Ad Hoc, o modo infraestrutura apresenta níveis hierárquicos e todas as comunicações passam obrigatoriamente por um AP, (não só as comunicações entre clientes de redes sem fios e com fios, mas também entre clientes de uma mesma rede sem fios), não existindo comunicação direta entre clientes. É importante salientar que os APs são fixos uma vez que estão conectados à infraestrutura da rede com fios, daí a origem do nome deste modo de funcionamento. Comparativamente ao modo de funcionamento Ad Hoc, o modo infraestrutura apresenta vantagens em termos de cobertura e escalabilidade, uma vez que os clientes podem comunicar indiretamente, ou seja, não necessitam de estar em alcance direto, e em termos de segurança, uma vez que apresenta um sistema

de gestão centralizado. Porém apresenta custos de implementação superiorres devido à necessidade de *hardware* extra, nomeadamente os APs. Este modo de funcionamento é o mais utilizado nas aplicações da IoT. É importante salientar que os modos de funcionamento não estão sempre diretamente refletidos na topologia da rede. Os modos podem ser vistos como uma definição básica de cada um dos nós, e não como uma caraterística de toda a infraestrutura [22].

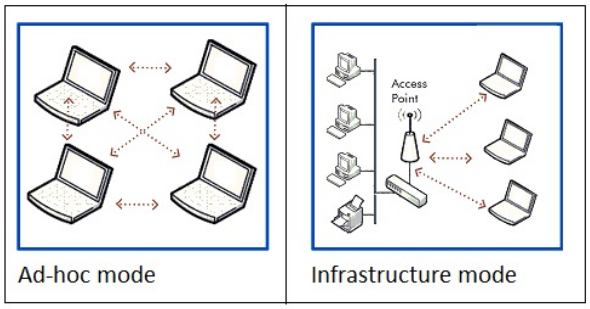


Figura 2.7: Modos de funcionamento de uma comunicação sem fios [23].

Dentro do modo de funcionamento infraestrutura e consoante a organização dos diferentes componentes presentes na rede, é possível obter-se diversas topologias, sendo as mais utilizadas a topologia em estrela e malha. A arquitetura em estrela é uma das estruturas mais comuns, por vezes também referida como a topologia *standard* para uma rede de comunicação sem fios. Nesta arquitetura, todos os clientes encontram-se conectados diretamente a um AP e por onde toda a informação deve obrigatoriamente passar. Isto resulta numa rede fiável e com uma capacidade de resposta rápida. Além disso, em caso de avaria de um nó, este é facilmente identificável e isolado. Por outro lado, o alcance da rede está diretamente relacionado com o alcance direto do dispositivo com que se pretende comunicar. Geralmente este tipo de topologia é expandida resultando numa topologia em árvore ou uma combinação com outro tipo de topologias [22][24].

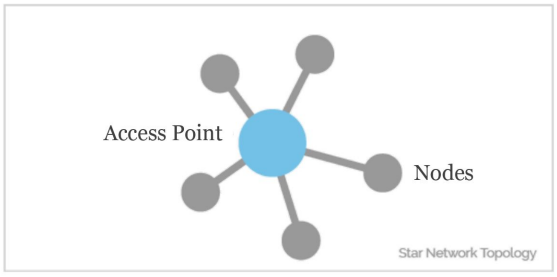


Figura 2.8: Topologia em estrela [24].

Em alternativa, a topologia em malha apresenta-se como uma opção interessante para ambientes urbanos e para situações onde uma infraestrutura central seja complicada de implementar. Esta topologia permite estruturar-se de duas formas diferentes, em malha completa ou em malha parcial. Numa estrutura em malha completa cada cliente encontra-se conectado diretamente a todos os outros. Já numa estrutura em malha parcial esta situação nem sempre acontece. Na prática a topologia em malha parcial é mais comum, uma vez que a distribuição dos diferentes componentes pode abranger uma área superior ao alcance individual de cada um dos nós impossibilitando a comunicação direta entre todos os clientes. Na topologia em malha é importante os clientes possuírem a capacidade de receber e reencaminhar informação. No caso de malha completa todos os nós devem possuir esta capacidade enquanto que em malha parcial apenas é necessário em alguns destes. Como vantagens da utilização desta topologia, esta permite suportar uma maior quantidade de tráfico, assim como é capaz de encontrar o caminho mais rápido e fiável, mesmo em caso de falha por parte de algum nó. Além disto é possível alterar a estrutura da rede, assim como expandi-la sem perturbar os nós já existentes. Por outro lado, como desvantagens, a existência de múltiplos caminhos aumenta a probabilidade de redundância na rede, da mesma maneira que aumenta a latência na comunicação devido à possibilidade da informação ter de passar por múltiplos nós até chegar ao seu destino [22][24].

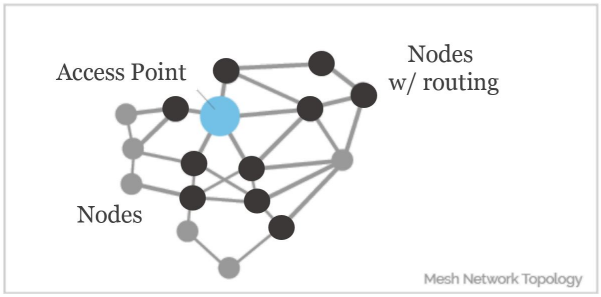


Figura 2.9: Topologia em malha parcial [24].

Existem diversas topologias com as suas vantagens e desvantagens, no entanto são as exigências da aplicação que ditarão a topologia adequada. É importante referir que, muitas das vezes, em situações reais, uma rede de comunicação sem fios é constituída por uma combinação de diferentes topologias.

2.3 Tecnologias de Comunicações Sem Fios para a Aplicação da Internet das Coisas - IoT

Com a enorme variedade de aplicações a escolha de uma comunicação para todas as soluções é difícil porém, existem atualmente no mercado diversas tecnologias disponíveis [25]. Cada uma destas tecnologias apresenta vantagens e desvantagens em diversos parâmetros tais como o custo, alcance, taxa de transmissão, consumo de energia, robustez a interferências, segurança, escalabilidade, isto é, a capacidade de uma rede já desenvolvida ser expandida, entre outras. Consoante estas caraterísticas, determinadas redes apresentam melhor capacidade para serem aplicadas em determinados cenários e ambientes.

Os diversos tipos de comunicação sem fios podem ser classificados consoante o alcance. A análise dos diferentes sistemas de comunicação terá por base a Figura 2.10. Nesta, é possível observar a existência de três grupos, WPAN, WLAN e WWAN, que serão abordados mais à frente, assim como algumas tecnologias presentes em cada um destes [17].

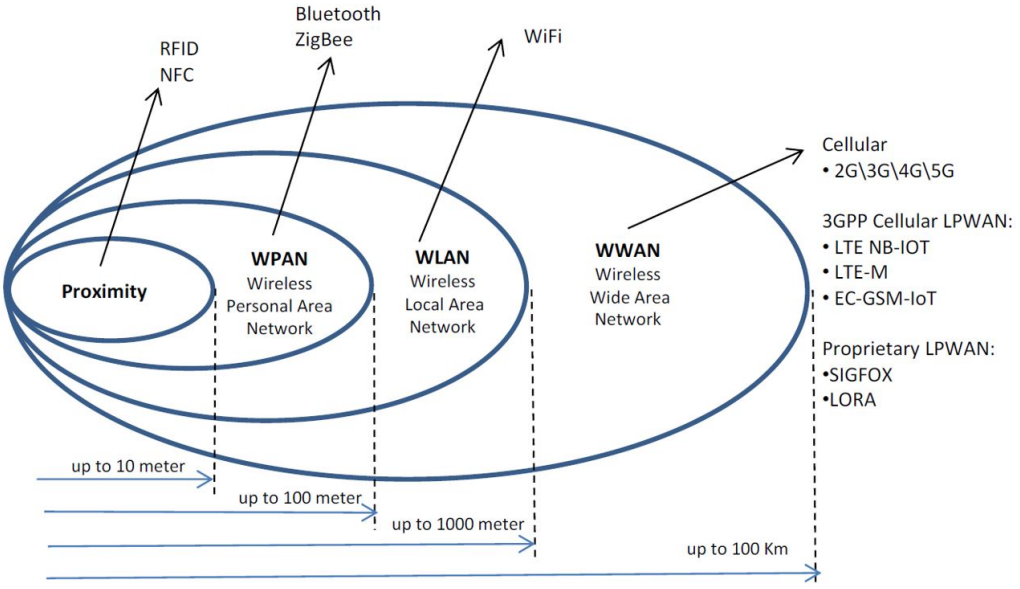


Figura 2.10: Classificação dos sistemas de comunicação sem fios [17].

Na Figura 2.11 é possível observar um gráfico com algumas das diferentes tecnologias de comunicação sem fios tipicamente utilizadas na IoT. Nesta figura estas tecnologias são classificadas segundo a taxa de transmissão, consumo de energia, alcance e custo [10]. Através de uma análise geral da Figura 2.11 é possível dividir as comunicações em dois grupos consoante o alcance, as comunicações com um alcance inferior a 100 metros como é o caso do RFID, Bluetooth, BLE, Zigbee, e as comunicações com um alcance superior a 100 metros que correspondem às comunicações móveis e às LPWAN. Existe uma excepção que é o Wi-Fi que, na generalidade dos casos, possui um alcance inferior a 100 metros porém, em determinadas situações, o seu alcance pode ser aumentado ultrapassando assim a barreira dos 100 metros. A partir da Figura 2.11 pode-se também observar que o RFID é a tecnologia com menor alcance, apresentando na maioria dos casos um alcance inferior a 10 metros, originando por vezes um outro grupo. Existem outros dois aspetos importantes que é possível retirar desta figura, o primeiro é o facto de quando maior for a taxa de transmissão maior será o consumo energético, o segundo é que quando maior for a taxa de transmissão e o alcance da tecnologia maiores serão os custos associadas a esta.

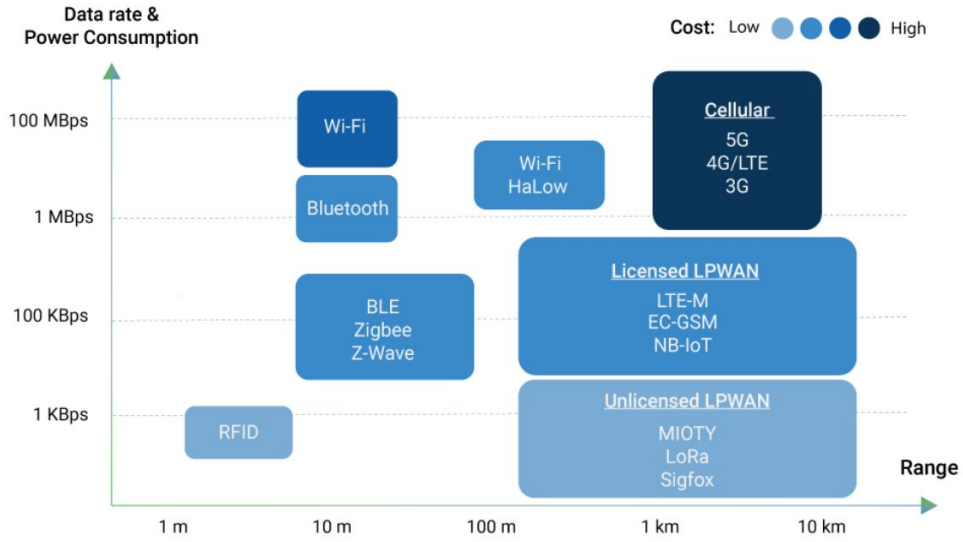


Figura 2.11: Tecnologias de comunicação sem fios [26].

A partir das Figuras 2.10 e 2.11 é possível obter um panorama geral da classificação das diferentes tecnologias existentes atualmente consoante o seu alcance, bem como a distinção entre estas em relação a alguns parâmetros tais como o consumo energético, taxa de transmissão e custo. Nos subcapítulos seguintes serão abordadas estas tecnologias, no entanto apenas de uma forma geral, uma vez que nem todas estas são o objetivo principal do trabalho.

2.3.1 Wireless Personal Area Network - WPAN

As *Wireless Personal Area Network* (WPAN) são redes essencialmente caraterizadas pelo seu curto alcance, tipicamente inferior a 100 metros [27]. O Bluetooth e o Zigbee são exemplos de tecnologias utilizadas neste tipo de redes. As WPAN possuem ainda um subgrupo, designado na Figura 2.10 de *Proximity*, que consiste em comunicações de alcance muito baixo, também designado de *contact range*, "distância de contacto", geralmente inferiores a 10 metros. Para estas distâncias são geralmente utilizadas tecnologias como o *Radio Frequency Identification* (RFID) e o *Near Field Communication* (NFC), no entanto apenas a primeira será referida, uma vez que o NFC é uma derivação da tecnologia RFID.

RFID é a tecnologia que apresenta o menor alcance, assim como uma baixa taxa de transmissão e consumo de energia quando comparada com as restantes. Esta é tipicamente utilizada em aplicações com um alcance inferior a 10 metros no entanto, o tipo de RFID *tag* utilizada e a frequência em que opera possui influência direta no alcance que, por vezes, pode atingir algumas dezenas de metros. A tecnologia RFID utiliza ondas de rádio para transmitir pequenas quantidades de dados de uma RFID *tag* para um leitor dentro de um determinado alcance. Existem dois tipos de RFID *tags*, as ativas, que possuem um bateria interna e tipicamente permitem processos de escrita e de leitura, e as passivas, que não possuem bateria no entanto recebem energia através de ondas eletromagnéticas fornecidas pelo leitor. A Figura 2.12 representa o princípio básico de funcionamento da tecnologia RFID, mas neste exemplo são utilizadas *tags* passivas. De um forma simplificada, quando se pretende efetuar uma leitura numa *tag* é gerado um campo magnético na antena do leitor RFID, este fornece energia suficiente à *tag* para que esta envie os dados da *tag* para o leitor para que posteriormente possam ser utilizados pela aplicação [28]. Os sistemas RFID podem operar a diferentes frequências possuindo determinadas particularidades. Consoante a frequência pode variar a distância máxima possível para efetuar a leitura, e consequentes necessidades energéticas e de desempenho. A área da logística e gestão de recursos foi a que mais tirou partido desta tecnologia. As empresas através da colocação de RFID *tags* nos seus produtos e equipamentos permitem controlar o seu inventário em tempo real e deste modo efetuarem uma melhor gestão do *stock* e da produção. Porém, muitas outras áreas têm usufruído desta tecnologia como por exemplo a área agrícola, da saúde, dos transportes, revenda de produtos e segurança. Atualmente existem diversas aplicações para esta tecnologia, no entanto esta poderá ser aplicada em muitas outras situações e cenários no futuro que nos dias de hoje ainda não se pensou que poderia ser aplicada [26][29][30].

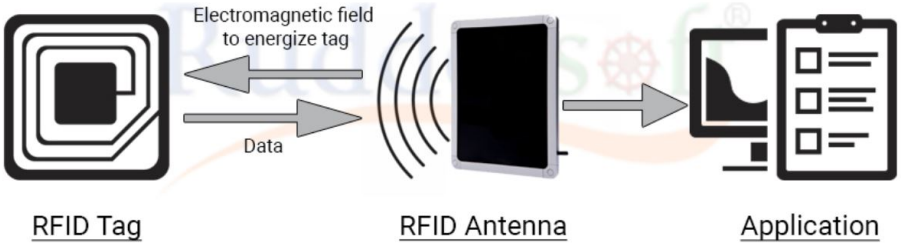


Figura 2.12: Princípio básico de funcionamento da tecnologia RFID [31].

O Bluetooth é uma comunicação sem fios de curto alcance bastante conhecida. Esta tecnologia possui também uma versão de baixo consumo, designada de Bluetooth Low-Energy (BLE) ou Bluetooth Smart, que foi especificamente otimizada para a utilização em aplicações da IoT. Esta opera globalmente a uma frequência de 2.4 GHz. Além disso, permite taxas de transmissão que variam entre os 125 kbps e os 2 Mbps e um alcance de cerca de 100 metros. Comparativamente ao Bluetooth tradicional, o BLE apresenta um maior alcance, porém possui uma menor taxa de transmissão e consumo energético. Isto deve-se ao facto de os dados serem enviados rapidamente e em pequenas quantidades e quando o dispositivo não necessita de enviar mais, este entra em "*sleep mode*" diminuindo assim o seu consumo energético. Atualmente o BLE já se encontra amplamente integrado em equipamentos médicos (sistemas vestíveis) e de *fitness*, bem como em equipamentos domésticos inteligentes [26][32].

O Zigbee, assim como BLE é um sistema de comunicação sem fios de baixo alcance e consumo energético. Este opera a uma frequência de 2.4 GHz e permite taxas de transmissão de 250 kbps. Geralmente é utilizado numa topologia em malha de modo a conseguir aumentar o seu alcance podendo ser utilizado para aplicações de médio alcance, entre os 10 e 100 metros. Comparado com as *Low Power Wide Area Network* (LPWAN), que serão abordadas mais à frente, neste tipo de alcance, permitem uma taxa de transmissão superior a estas, porém, devido à configuração em malha, o consumo energético é significativamente superior. Até ao aparecimento das tecnologias LPWAN, o Zigbee com topologia em malha era bastante usado na IoT na área industrial para soluções de monitorização remota. No entanto não é a solução ideal para instalações geograficamente dispersas, uma vez que há medida que o tamanho da rede aumenta, a dificuldade na sua escalabilidade também aumenta, implicando configurações e uma gestão da rede mais complexa [26][33].

2.3.2 Wireless Local Area Network - WLAN

As redes sem fios de médio alcance são designadas de *Wireless Local Area Network* (WLAN) e caraterizam-se por ter um alcance inferior a 1 km. O *Wireless Fidelity* (Wi-Fi) é uma das WLAN mais conhecidas e utilizadas atualmente, não só no dia a dia, mas também na IoT. Este pode operar nas frequências dos 2.4 GHz e 5 GHz e permite taxas de transmissão entre os 2 Mbps e os 1730 Mbps. O alcance típico do Wi-Fi varia entre os 50 a 100 metros.

Porém, apesar da versatilidade que o Wi-Fi possui, na generalidade dos casos esta não tem uma aplicabilidade direta, isto é, não controla diretamente sensores ou atuadores devido aos seus problemas de cobertura, no entanto é bastante utilizada como uma rede secundária responsável por enviar os dados de um *hub* central (que recebe os dados dos diferentes sensores) e enviá-los para a cloud. Isto deve-se às características que esta rede possui, especialmente pela boa taxa de transmissão. Por outro lado, além da baixa cobertura, o Wi-Fi possui outras limitações tais como um alto consumo energético, custos superiores relativamente a maioria das outras tecnologias e dificuldades em termos de escalabilidade que levam a que este tipo de tecnologia não seja, geralmente, adotada diretamente em soluções industriais ou comerciais [26].

2.3.3 Wireless Wide Area Network - WWAN

Para alcances superiores a 1 km as redes de comunicação sem fios são denominadas de *Wireless Wide Area Network* (WWAN). Neste tipo de redes são geralmente utilizadas dois tipos de tecnologias de comunicação sem fios, as redes de comunicação móveis e as *Low Power Wide Area Network* (LPWAN). Dentro das LPWAN estas podem ainda se dividir em dois grupos consoante a banda de frequência utilizada e referem-se como LPWAN licenciada e LPWAN não licenciada, isto é, se operam em bandas de frequência isentas ou não de algum tipo de taxa de utilização.

Os sistemas de comunicação móveis possui um alcance de 200 km e operam nas frequências dos 900, 1800, 1900, 2100 MHz. Os valores para a taxa de transmissão variam entre 50 Mbps e 300 Mbps. Comparativamente às restantes tecnologias apresentadas, as redes de comunicação móveis oferecem uma largura de banda fiável e uma taxa de transmissão superior, com exceção do Wi-Fi. Por outro lado, apresentam um grande consumo energético tornando este tipo tecnologia inviável para aplicações da IoT que necessitam de uma bateria como fonte de energia. No entanto são uma boa escolha para algumas áreas da IoT como por exemplo a conectividade entre veículos e em aplicações de gestão de frotas. Dentro dos diferentes standards de redes de comunicações móveis destacam-se a *Global System for Mobile Communications* (GSM), 3G e 4G, sendo estas últimas por vezes designadas respetivamente de *Universal Mobile*

Telecommunications System (UMTS) e *Long Term Evolution* (LTE) [17][26]. Com a introdução do 5G, esta fornecerá três grandes benefícios comparativamente ao sistemas de redes móveis referidos anteriormente, nomeadamente uma maior largura de banda, uma qualidade de serviço superior em termos de fiabilidade e latência e possibilitará a sua aplicação numa maior variedade de dispositivos. Devido a estas caraterísticas é esperado que o 5G possua um papel importante no desenvolvimento de diversas áreas da IoT.

LPWAN é um termo genérico para um grupo de tecnologias que possui um conjunto de características chave. Estas devem permitir o estabelecimento de comunicações de longo alcance, mas com um baixo consumo energético, de modo a permitir uma longa duração da bateria (tipicamente superior a 10 anos). Além disso devem ser sistemas de fácil implementação e de baixo custo. Associado a isto existe sempre a desvantagem de baixas taxas de transmissão. Estas tecnologias vêm complementar as redes móveis e as redes de curto alcance de modo a permitir comunicações a longas distâncias com menores custos e com melhor eficiência energética. Na sua generalidade estas tecnologias apresentam uma pequena largura de banda e operam no espectro de frequências *Industrial, Scientific and Medical* (ISM) isenta de licença. As que operam neste espetro de frequências são designadas de LPWAN não licenciadas. Este trabalho incide sobretudo neste tipo de redes. *Long Range* (LoRa) e *SigFox* são dois exemplos de tecnologias típicas neste tipo de redes. A tecnologia LoRa permite uma cobertura de cerca de 15 km e com taxas de transmissão até 50 kbps, enquanto que o Sigfox permite um alcance de 50 km mas uma taxa de transmissão de apenas 100 bps.

No entanto existem outras tecnologias que se baseiam no modelo de comunicação e infraestruturas das redes móveis, uma vez que têm por base as mesmas tecnologias referidas anteriormente (GSM, 3G, 4G) e são designadas de LPWAN licenciadas, uma vez que operam em bandas de frequência reservadas. As tecnologias LTE NB-IoT, LTE-M e EC-GSM, que tem por base o sistema de redes móveis LTE e GSM, são alguns exemplos de LPWAN licenciadas [34].

2.3.4 Resumo

A Tabela 2.2 apresenta um resumo das diferentes tecnologias abordadas ao longo do capítulo 2.3, com algumas das caraterísticas mais importantes, tais como o alcance, taxa de transmissão, frequência de operação e o consumo energético. A tecnologia RFID não foi incluida nesta tabela uma vez que alguns dos parâmetros podem variar bastante consoante a frequência em que operam e o equipamento utilizado.

Tabela 2.2: Tabela resumo das diferentes tecnologias de comunicação sem fios.

Tecnologia	**Alcance (Típico)**	**Taxa de transmissão**	**Frequência**	**Consumo energético**
BLE	<100 m	125 - 2000 Mbps	2.4 GHz	Baixo
Zigbee	<100 m	250 kbps	2.4 Ghz	Baixo
Wi-Fi	100 m	2 -1730 Mbps	2.4 GHz ou 5 GHz	Alto
Sist. Móveis	<200 km	50 - 300 Mbps	900/1800/1900/2100 MHz	Alto
LoRa	<15 km	50 kbps	ISM	Baixo
Sigfox	<50 km	100 bps	ISM	Baixo

Uma vez que para implementação do sistema se pretende utilizar uma tecnologia de comunicação sem fios de longo alcance e de baixo consumo, de todas as tecnologias apresentadas, as que se enquadram melhor neste cenário são a LoRa e a Sigfox. Os sistemas móveis poderiam ser uma opção devido ao seu alcance, porém apresentam um consumo energético alto. As restantes tecnologias são desfavoráveis, visto que apresentam alcances baixos.

2.4 Protocolos de Comunicação - IoT

No conceito da IoT a comunicação pode ser dividida em dois grandes grupos: a comunicação entre dois dispositivos e comunicação entre um dispositivo e a *Internet*. As comunicações entre dispositivos, nomeadamente entre nós ou *gateways*, já foram referidas no capítulo 2.3. Deste modo, com este capítulo pretende-se abordar alguns dos protocolos utilizados para a comunicação entre um dispositivo e a *Internet*. Todos os protocolos apresentam vantagens e desvantagens uma vez que foram desenvolvidas com determinadas aplicações em mente, no entanto todos eles têm como objetivo reduzir a largura de banda utilizada e o número de *bytes* a transmitir por trama. Nos secções seguintes serão abordados de uma forma breve alguns dos protocolos mais comuns referindo as vantagens e desvantagens de cada um destes.

2.4.1 HTTP/2

O *Hypertext Transfer Protocol* (HTTP) é um protocolo desenvolvido para a troca ou transferência de hipertexto. O (HTTP)/2 é o resultado de uma revisão moderna do HTTP original que tem como objetivo resolver alguns dos problemas existentes na versão mais antiga, HTTP/1.1, mantendo simultâneamente a compatibilidade com esta. Nesta nova versão as características fundamentais presentes no HTTP/1.1 são mantidas, porém possibilita um uso mais eficiente dos recursos de rede ao possibilitar a compressão dos campos de cabeçalho, que nesta versão são em binário em vez de texto, e ao permitir intercalar mensagens de solicitação e resposta na mesma conexão. Além disto, adicionou um mecanismo de controlo de fluxo que permite a priorização de solicitações, permitindo assim que as mais importantes sejam concluídas mais rapidamente, resultando numa melhoria no desempenho. É importante referir que o HTTP tem por base o *Transmission Control Protocol* (TCP) e que utiliza *Transport Layer Security* (TLS) como mecanismo de segurança [35].

A grande dimensão dos cabeçalhos impostos pelo protocolo HTTP podem, por vezes, causar *overhead* na comunicação tornando a sua utilização inviável em dispositivos restringidos a nível de *hardware* e alimentação.

Por outro lado, uma das vantagens da utilização do HTTP é o facto de este ser praticamente suportado em qualquer lugar o que permite uma maior compatibilidade com outras aplicações. Além disso é um protocolo muito utilizado globalmente, existindo uma comunidade com uma enorme quantidade de informação disponível acerca do mesmo [35][36].

2.4.2 MQTT

Message Queue Telemetry Transport (MQTT), criado pela IBM, é um protocolo leve desenvolvido essencialmente para ser aplicado na IoT. Este é um protocolo assíncrono, que utiliza uma arquitetura *publish/subscribe* e que tem por base o protocolo de transporte TCP.

A arquitetura *publish/subscribe* distingue-se pela existência de um intermediário denominado de *broker* que é responsável por gerir todas as mensagens. Neste caso, os clientes podem subscrever ou publicar mensagens em diferentes tópicos, tendo o *broker* a responsabilidade de reencaminhar as mensagens para os respetivos destinatários. Este protocolo apresenta três níveis de *Quality of Service* (QoS) que permitem garantir a entrega das mensagens entre um cliente e o *broker*. A utilização do QoS tem como objetivo facilitar a comunicação em redes pouco fiáveis, sendo uma vantagem do protocolo MQTT. Além disso, e ao contrário do protocolo HTTP, o MQTT permite cabeçalhos de menores dimensões tornando-o adequado para redes com uma largura de banda pequena e para dispositivos restringidos a nível de *hardware*. Uma desvantagem da utilização da arquitetura *publish/subscribe* é o facto do *broker* apresentar influencia direta na escalabilidade e desempenho da rede, uma vez que à medida que mais clientes se conectam, mais sobrecarregada esta fica. Deste modo, a rede apenas pode crescer consoante a capacidade do *broker* [37].

2.4.3 CoAP

O *Constrained Application Protocol* (CoAP), assim como HTTP é um protocolo desenvolvido para a troca ou transferência de hipertexto. Porém, ao contrário do HTTP, este foi desenvolvido para ser aplicado em dispositivos restringidos a nível de hardware. Os pacotes de dados gerados pelo CoAP são menores relativamente aos do HTTP. Isto deve-se ao facto de o CoAP operar sobre o protocolo de transporte *User Datagram Protocol* (UDP), permitindo assim reduzir o *overhead* em cada comunicação causado pelas dimensões do cabeçalho. Por outro lado, é necessário a implementação de mecanismos de controlo de fluxo e erros uma vez que as comunicações UDP não são tão fiáveis como as TCP. De forma a garantir segurança durante a troca de mensagens entre cliente e servidor, o CoAP usa o protocolo DTLS (*Datagram Transport Layer Security*), que é baseado no TLS mas sobre UDP em vez do TCP. A grande desvantagem deste protocolo é o facto de não suportar comunicação *one-to-many*, isto é, não é possível enviar uma mensagem para múltiplos destinatários em simultâneo [38][39].

2.4.4 AMQP

Advanced Message Queueing Protocol (AMQP) é um protocolo desenvolvido para a troca de mensagens. Relativamente aos protocolos já referidos, este apresenta bastantes semelhanças com o MQTT, no entanto possui as suas particularidades. Assim como o MQTT, o AMQP opera sobre o fiável protocolo da camada de transporte TCP, usufruindo também do TLS para a segurança nas transmissões. Porém, o AMQP não utiliza uma arquitetura *publish/subscribe* pura como o MQTT permitindo no entanto operar como uma. O AMQP possui a particularidade de permitir a colocação de mensagens em filas de mensagens consoante a sua prioridade. Além disso possui a capacidade de as armazenar, tornando este protocolo mais fiável no caso de existir interrupções na rede. Assim como o MQTT este apresente três níveis de QoS. A desvantagem deste protocolo é o facto de não ser tão leve quando comparado com o MQTT uma vez que apresenta um maior *overhead* devido ao aumento de funcionalidades que este possui [38][40].

2.4.5 Resumo

A Tabela 2.3 pretende resumir os aspetos mais importantes dos protocolos de comunicação abordados neste capítulo.

Um protocolo leve e pouco exigente a nível de *hardware* é o principal objetivo na escolha do protocolo a utilizar. Perante isto, os protocolos HTTP/2 e AMQP não são os mais adequados uma vez que são mais pesados relativamente aos outros dois. Tanto o CoAP como o MQTT enquadram-se no principal factor, no entanto a escolha final recaí sobre o MQTT uma vez que este suporta comunicações *one-to-many*, opera sobre o fiável protocolo da camada de transporte TCP e apresenta uma maior simplicidade em termos de implementação.

Tabela 2.3: Tabela resumo dos diferentes protocolos de comunicação. Adaptado de [35].

Protocol	**HTTPv2**	**CoAP**	**MQTT**	**AMQP**
Architecture Style	Client/servers model	Client/server model RESTful	Brokered style	Brokered style
Transport	TCP	UDP	TCP	TCP
Messaging	Request/ Response (Supports multiplexing)	Request/ Response	Publish/ Subscribe	Publish/ Subscribe
Header	Binary (header compression)	4Byte Binary-based	Fixed length of 2Byte	8Byte
Service levels (QoS)	Priority mechanism of streams	Confirmable or nonconfirmable messages	Three quality of service settings	Three quality of service settings
Data distribution	One-to-one and one-to-many	One-to-one	One-to-one and one-to-many	One-to-one and one-to-many
Security	Requires TLS version 1.2 or higher	Typically based on SSL or TLS	Simple User/ Password Authentication, SSL for data encryption	SASL authentication, TLS for data encryption

2.5 Sistemas *Healthcare*

Cuidados médicos e de saúde representam uma das áreas da IoT com um grande potencial para se desenvolver no futuro. Apesar de atualmente não ser um dos tópicos dominantes, os projetos e serviços realizados nesta área têm vindo a aumentar nos últimos anos. Prevê-se que esta tendência de crescimento se mantenha no futuro e, como é possível observar na Figura 2.13, é esperado que em 2025 esta possua uma quota no mercado de cerca de 41 %, tornando-se na área de maior impacto económico [2].

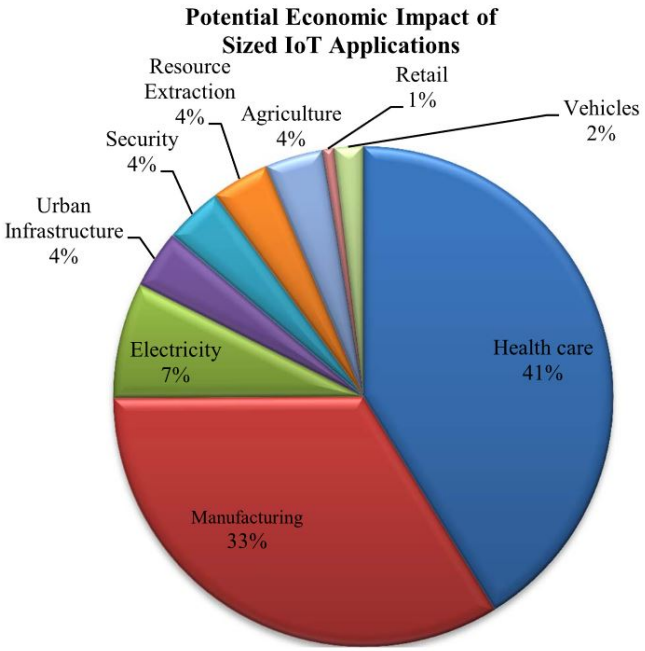


Figura 2.13: Previsão da quota do mercado das principais áreas de aplicação da IoT [2].

2.5.1 Tecnologias Inerentes aos Sistemas *Healthcare*

A Figura 2.14 representa o princípio básico de funcionamento de um sistema *healthcare* baseado na filosofia da IoT. Nestes sistemas a informação é obtido através de um ou vários sensores médicos, podendo por vezes formar redes de sensores também designadas de *Wireless Body Area Netowrks* (WBANs). Este dados são posteriormente enviados e armazenados, normalmente em servidores *cloud*, para que possam ser acedidos e analisados pelos médicos ou por utilizadores autorizados. No entanto, apesar de a Figura 2.14 representar um esquema simplificado destes sistemas é possível observar que, desde a forma de captura da

informação até à análise da mesma, estes sistemas podem envolver um diverso número de tecnologias. Dentro destas é possível salientar as de maior importância tais como, sensores médicos, comunicações sem fios, *big data* e *cloud* [41][42].

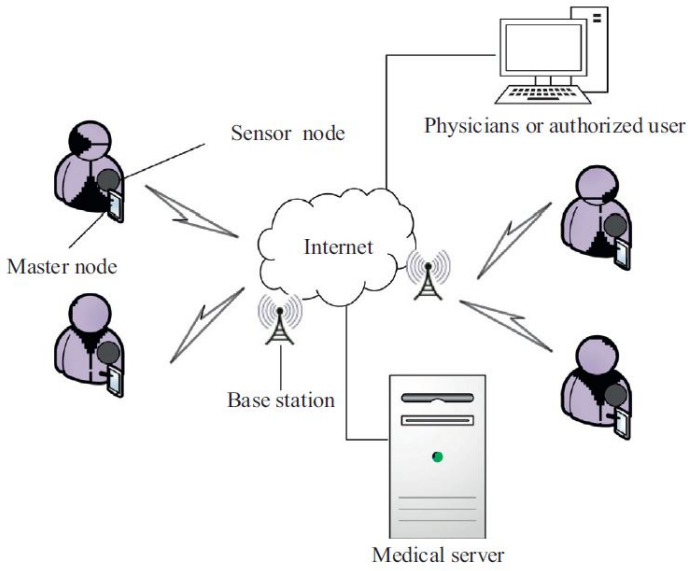


Figura 2.14: Esquema simplificado de um sistema *healthcare* baseado na filosofia da IoT [43].

- **Sensores médicos e comunicações sem fios:** Os sensores desempenham um papel fundamental nos sistemas *healthcare*, pois são estes que irão permitir adquirir grande parte da informação necessária para as diferentes aplicações e serviços. Estes sensores associados a algumas das comunicações sem fios abordadas anteriormente possibilitam o surgimento de sistemas vestíveis. Esta tecnologia possui grande relevância nos sistemas *healthcare*, uma vez que facilita a aquisição de dados, mesmo durante longos períodos de tempo, minimizando os transtornos causados ao utilizador. Na tabela 2.4 é apresentada uma lista dos sensores típicos utilizados nesta área e a sua respetiva função.
- **Big Data:** Os sensores recolhem uma grande quantidade de dados e toda esta informação necessita de ser analisada. *Big data*, representa aqui um papel fundamental no fornecimento de ferramentas que permitem aumentar a eficiência no correto diagnóstico de problemas e nos métodos de monitorização.
- **Cloud Computing:** A integração da *cloud computing* nos sistemas *healthcare* baseados na filosofia da IoT permite facilitar o acesso a recursos comparti-

lhados, oferecendo serviços mediante a sua solicitação pela rede e executando operações para atender as várias necessidades.

Tabela 2.4: Sensores médicos tipicamente utilizados em sistemas vestíveis [41].

Sensor	***Função***
Ritmo Cardíaco	Para detectar a frequência cardíaca (e consequentemente a variabilidade da mesma de modo a detetar problemas, como por exemplo arritmias)
ECG (Eletrocardiografia)	Para medir a atividade elétrica do coração, e deste modo obter informações essenciais sobre o estado do coração em função do movimento do seus músculos
EMG (Eletromiografia)	Para medir o sinal elétrico provocado pela atividade muscular. Permite o reconhecimento de gestos, detecção de doenças neuromusculares, etc.
EEG (Eletroencefalografia)	Para capturar sinais elétricos no cérebro, permitindo representar a atividade presente no mesmo.
Pressão sanguínea	Para medir a pressão arterial sistólica e diastólica.
Taxa de respiração	Para medir o taxa de respiração
SpO2	Permite medir a quantidade de oxigénio dissolvido no sangue.
Condutividade da pele	Para medir a condutividade da pele de forma a obter o nível de humidade da pele. Permite ainda detectar excitação psicológica ou fisiológica.
GSR (Galvanic Skin Response)	Permite medir o nível de transpiração.
Co2	Permite medir a quantidade de dióxido de carbono presente no ar.
Glicosímetro	Sensores que registam os níveis de glicose continuamente no tempo.
Sensores de movimento	Permite detetar/seguir movimentos.
Sensores de stress	Para medir o nível de stress através da medição das mudanças de pressão na parte inferior do pé.
Acelerómetro	Permite medir a energia despendida pelo utilizador.

2.5.2 Sistemas *Healthcare* - Serviços e Aplicações

Em relação aos serviços e aplicações na área dos cuidados médicos e de saúde baseados na filosofia da IoT, estes podem abranger uma enorme variedade de aplicações, desde a gestão particular da saúde e da atividade fisíca, até à supervisão de doenças crónicas, cuidados na área da pediatria e de idosos, entre outras. No sentido de facilitar o entendimento deste tópico extenso, este é geralmente dividido em dois grandes grupos: serviços e aplicações. É importante salientar que esta classificação é baseada no estado atual dos serviços e aplicações desenvolvidos e em desenvolvimento, no entanto com o aparecimento de novas tecnologias esta classificação poderá sofrer alterações através da modificação ou adição de serviços ou aplicações. Esta classificação encontra-se apresentada na Figura 2.15. Devido à grande amplitude deste tópico, neste trabalho apenas será abordado parte deste, nomeadamente o serviço relacionado com os sistemas vestíveis e algumas aplicações de maior interesse para o mesmo [44].

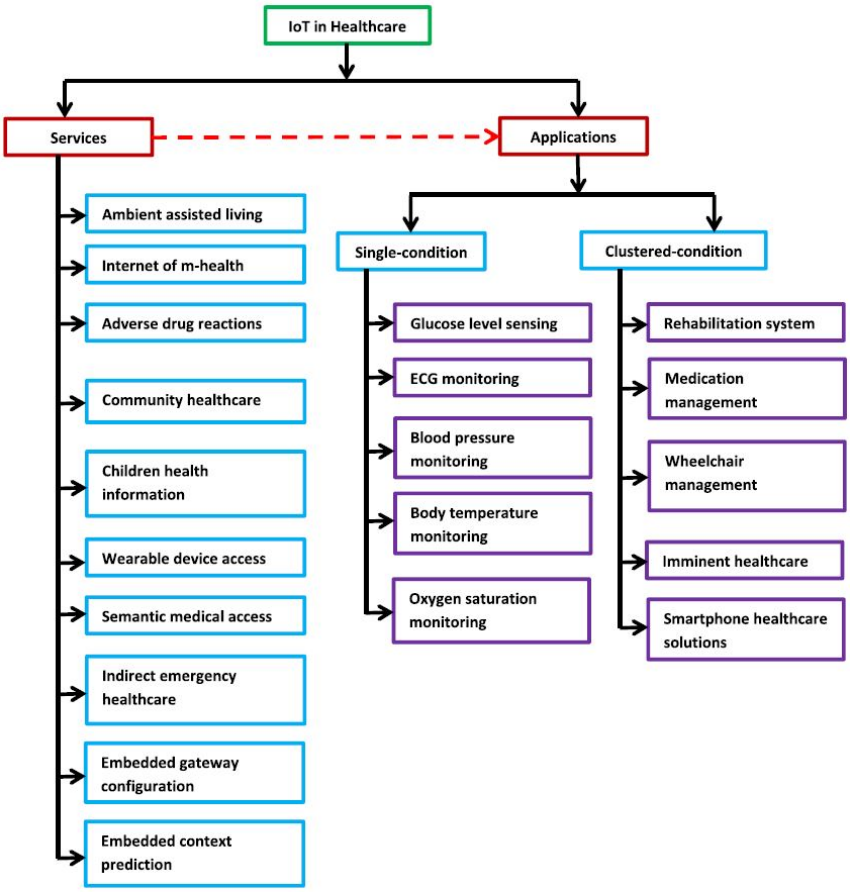


Figura 2.15: Classificação dos sistemas *healthcare* [44].

2.5.2.1 Serviços *Healthcare*

Prevê-se que a IoT possibilite uma variedade de serviços, os quais permitirão desenvolver um conjunto de soluções a partir destes. Um serviço pode ser visto como um bloco genérico que tem como o objetivo servir de base para um conjunto de aplicações mais especificas. Por outras palavras, cada serviço consiste num conjunto de protocolos inerentes à IoT, assim como outros protocolos necessários e que servirão de alicerce para o desenvolvimento de diferentes aplicações. No entanto, dentro do mesmo serviço, pode ser necessário ligeiras modificações de modo otimizar o funcionamento de cada uma das aplicações. Como é possível observar pelo diagrama apresentado na Figura 2.15 existem atualmente diversos serviços no entanto apenas o serviço referente aos sistemas vestíveis será abordado.

Nos últimos anos diversos sensores não intrusivos foram desenvolvidos para um grande variedade de aplicações médicas. Muitos destes podem ser integrados em redes de sensores sem fios (WSN) e com potencial para serem conjugados com a IoT. Após isto, a interligação destes sensores com produtos vestíveis é aparente, no entanto esta interligação levanta diversos desafios tanto para investigadores como fabricantes de produtos, daí a necessidade da existência de um serviço ligado a estes sistemas vestíveis, designado de *Wearable Device Access* (WDA) [43][44]. No entanto, os sistemas vestíveis encontram-se numa fase inicial do seu ciclo de vida e, neste momento, este serviço apenas se trata de um conceito, porém espera-se este venha a ser adotado no futuro de modo a facilitar o desenvolvimento de aplicações nesta área.

2.5.2.2 Aplicações *Healthcare*

Em relação às aplicações, estas podem ser visto como produtos que servem de solução para determinados problemas. No entanto, em determinadas situações, a distinção clara entre serviços e aplicações pode ser ambígua. Porém, os serviços podem ser vistos como ferramentas para o desenvolvimento de aplicações (*developer-centric*), enquanto que as aplicações são um produto final dirigido a utilizadores/pacientes (*user-centric*). Baseado na estrutura da Figura 2.15, estas aplicações podem ser divididas em dois grupos, designados de *single condition* ou *clustered condition*, isto é, as aplicações *single condition* tem como objetivo monitorizar/controlar apenas um parâmetro, enquanto que as aplicações *clustered condition* são desenvolvidas para resolver problemas mais complexos e que por norma envolvem múltiplos parâmetros. Neste trabalho será atribuída maior ênfase às aplicações *single condition*.

Grande parte das aplicações consiste na monitorização de um único parâmetro. A monitorização destes parâmetros permitirá detetar antecipadamente possíveis problemas e desta forma tomar medidas preventivas de modo a di-

minuir o seu surgimento no futuro. Estas são tipicamente divididas em cinco grupos diferentes e encontram-se listadas abaixo [43][44]:

- **Níveis de glicose:** Diabetes é um grupo de doenças metabólicas provocadas por níveis altos de glicose no sangue durante longos períodos de tempo. A monitorização permanente dos níveis de glicose no sangue irá permitir identificar padrões individuais e deste modo ajudar no planeamento das refeições, atividades e medicação.

- **Sinal ECG:** Através da análise de um eletrocardiograma (ECG), que reflete a atividade elétrica do coração obtida através de um eletrocardiografo, é possível obter várias informações acerca deste orgão, desde o simples batimento cardíaco até ao diagnóstico de diferentes problemas associados a este como é o caso de arritmias. A aplicação da IoT para monitorização do sinal ECG pode trazer inúmeras vantagens.

- **Pressão sanguínea, temperatura corporal e saturação do oxigénio:** Estes três parâmetros encontram-se agrupados num só tópico pois o objetivo da sua monitorização é bastante semelhante. Apesar disto, a monitorização destes parâmetros representam um papel fundamental nos serviços médicos, uma vez que qualquer anomalia nestes pode perturbar a homeostasia corporal, sendo um indicador de possíveis problemas no organismo.

Atualmente existem no mercado diversas soluções que permitem monitorizar estes parâmetros. Um exemplo é o MySignals, uma plataforma de desenvolvimento para aplicações da IoT no segmento da saúde. Esta é uma plataforma modular que permite medir mais de 15 parâmetros através de diferentes sensores disponiveis pela própria MySignals ou outros sensores que se encontrem disponíveis no mercado compatíveis com esta plataforma. A MySignals dispõe de sensores para a medição da temperatura corporal, níveis de glicose, pressão sanguínea, saturação de oxigénio no sangue, posição corporal entre outros. Para além destes é possível encontrar no mercado outros sensores compatíveis com esta plataforma que permitem capturar outros parâmetros, como por exemplo o sinal ECG e EMG [45]. Esta plataforma possui dois tipo de comunicações sem fios, BLE e Wi-Fi, que permitem enviar os dados recolhidos diretamente para a *cloud* ou para o *smartphone* e posteriormente para a *cloud*. Todas as informações recolhidas podem ser posteriormente analisadas no *smartphone*, no computador ou no próprio equipamento. A Figura 2.16 representa a arquitetura do sistema MySignals. É importante salientar que esta plataforma não é um produto final, sendo desenvolvida intencionalmente como um produto base para investigadores e desenvolvedores de produtos [46].

Figura 2.16: Arquitetura do sistema MySignals [46].

A iHealth é outra empresa que dispõe de produtos na área dos cuidados médicos de saúde. Equipamentos para a medição da pressão arterial, dos níveis de glicose e da saturação de oxigénio são alguns dos produtos baseados na filosofia da IoT que esta empresa desenvolveu. Todos estes dispositivos permitem a recolham de informação através de sensores e posteriormente enviam os dados para a *cloud*. Primeiramente a informação é enviada via *bluetooth* para o *smartphone* e de seguida é reencaminhada para a *cloud* utilizando o Wi-Fi. A informação pode ser consultada através de uma aplicação para *smartphone* ou numa página *web* onde é possível observar todas a medições efetuadas até ao momento atual e, caso seja necessário, possibilita a partilha de informação com um profissional de saúde. Além disto, a aplicação para *smartphone* gera notificações para alertar o utilizador para realizar uma nova leitura ou para tomar algum tipo de medicação [47].

Capítulo 3

Tecnologia LoRa

LoRa é uma abreviação de *Long Range* e refere-se a um sistema de comunicação de dados, sem fios e de longo alcance desenvolvida pela Semtech [48]. Este tipo de LPWAN foi desenvolvida com o objetivo de ser implementada em dispositivos com capacidades energéticas limitadas (por exemplo, que tenham como alimentação uma bateria), que apenas necessitem de enviar pequenas quantidades de dados a uma baixa frequência e que a comunicação possa tanto ser iniciada pelo próprio dispositivo (caso o dispositivo se trate de um sensor) ou por uma entidade externa (caso o dispositivo se trata de um atuador). As características desta tecnologia colocam-na numa posição interessante para aplicações de monitorização não só a nível industrial, mas também na área da saúde e ambiental [49]. Antes de avançar é necessário salientar que LoRa e LoRaWAN são dois conceitos diferentes dentro desta tecnologia. De uma forma sucinta e comparativamente ao modelo OSI, LoRa corresponde à camada física enquanto que LoRaWAN refere-se à camada de ligação de dados.

3.1 LoRa

LoRa corresponde à camada física e esta utiliza uma técnica de modelação rádio denominada de *Chirp Spread Spectrum* (CSS). Esta permite codificar informação através da variação linear da frequência dentro de uma largura de banda definida. Esta tem por base dois tipos de variações, as variações lineares ascendentes e descendentes designadas, respetivamente de *upchirp* e *downchirp*. Comparativamente à técnica de modulação mais comum para comunicações de baixo consumo e de longo alcance, *frequency shifting keying* (FSK), a modulação CSS mantém na mesma a baixa necessidade energética, porém permite aumentar significativamente o alcance. Esta técnica já é usada há algumas décadas em

comunicações militares e espaciais devido ao seu grande alcance e robustez a interferências, no entanto o LoRa é a primeira solução de baixo custo para uso comercial. Além disso opera nas bandas de frequência ISM, 433, 868 e 915 MHz, dependendo da região onde é aplicada [49][50][51]. De uma forma geral, a tecnologia LoRa adequa-se melhor a aplicações que necessitem de uma comunicação sem fios de baixo consumo energético, de longo alcance, bidirecionais e onde a transmissão de dados em tempo real não seja um fator crítico.

3.1.1 Parâmetros da Camada Física

A modelação LoRa permite a modificação de vários parâmetros de modo a obter-se diferentes configurações em termos de consumo energético e alcance. No entanto estes também influênciam o tempo de transmissão de dados, a velocidade de modelação e desmodulação e a resistência a interferências. Estes parâmetros são o *Spreading Factor* (SF), *Bandwidth* (BW) e o *Coding Rate* (CR) [52].

O *Spreading Factor* (SF) representa o número de *chips* por *symbol*, por outras palavras, o número de *chips* presentes num *chirp*. Um *symbol* representa um *chirp* e contém a informação que se pretende enviar. O SF permite definir dois valores, o número de *bits* que podem ser utilizados para codificar um *symbol* e o número de *chips* que um *symbol* pode conter, isto é, a gama de valores que cada *symbol* pode assumir. O número de *chips* é definido pela potência de base 2 do SF (2^{SF}). Por exemplo, se for utilizado um SF de 8 o número de *chips* por *symbol* será de 256 (2^8 = 256 chips/symbol), isto quer dizer que o valor decimal contido por *symbol* pode variar entre 0 e 255. É importante não confundir o conceito de *chirp* e *chip*. A modulação LoRa possibilita a utilização de 6 valores de SF, que variam entre 7 e 12. Na Tabela 3.1 é possível observar a relação entre o SF e o número de *chips* por *symbol*. O valor do SF pode variar entre 7 e 12 e a alteração deste tem implicações diretas no alcance e na taxa de transmissão. Quanto maior for o valor de SF maior será o alcance, no entanto menor será a taxa de transmissão, e vice-versa. Através da Figura 3.1 é possível observar que a cada incremento do SF a taxa de transmissão diminui para metade e, consequentemente, o tempo de transmissão aumenta para o dobro [53][52][54].

Tabela 3.1: Relação entre SF e o número de *chips* por *symbol* [52].

Spreading Factor (SF)	Chips/Symbol
7	128
8	256
9	512
10	1024
11	2048
12	4096

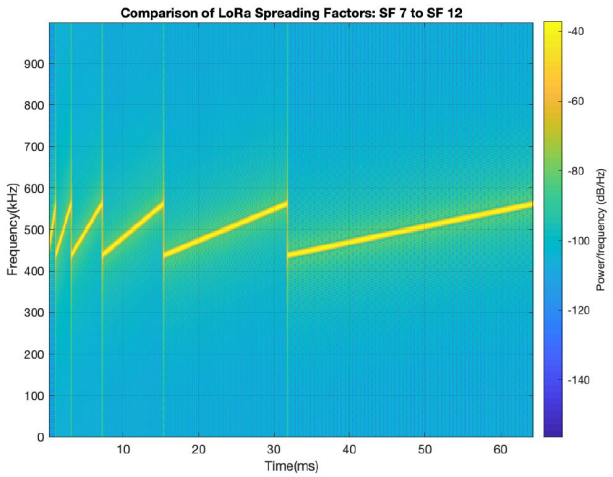


Figura 3.1: Relação entre os diferentes SF e o tempo de transmissão para o mesmo symbol [55].

A BW representa a diferença entre o limite superior e o inferior da banda de transmissão centrada na frequência portadora, *Carrier Frequency* (CF), que na Europa varia entre os 863 MHz e os 870 MHz. LoRa possibilita a escolha de três opções de BW, 125 MHz, 250 MHz e 500 MHz. Caso se pretenda otimizar a velocidade de transmissão é recomendado utilizar uma largura de banda de 500 MHz, caso se pretenda aumentar o alcance da comunicação é preferível utilizar um BW de 125 MHz [52].

O *Coding Rate* (CR) é outro parâmetro que é possível definir. Este parâmetro é utilizado para obter o *Cyclic Coding Rate* (CCR) e representa a proporção de *bits* transmitidos que efetivamente contém informação, os restantes servem apenas para controlo de erros. O CCR é obtido através da expressão 3.1, onde o CR pode assumir os valores 1, 2, 3 e 4. Quanto menor for o CR (maior o CCR), menor será

o tempo necessário para a transmissão de dados.

$$CCR = \frac{4}{4 + CR} \tag{3.1}$$

Com estes parâmetros é possível obter a taxa de transmissão nominal (R_b), em *bits* por segundo, através da expressão 3.2 [52].

$$R_b = SF \cdot \frac{BW}{2^{SF}} \cdot CR \tag{3.2}$$

A Figura 3.2 representa a variação da frequência no tempo de uma transmissão LoRa. Nesta é possível observar alguns dos parâmetros e conceitos abordados neste capítulo tais como a BW, o SF, um *upchirp* e *downchirp*, assim como um *symbol*.

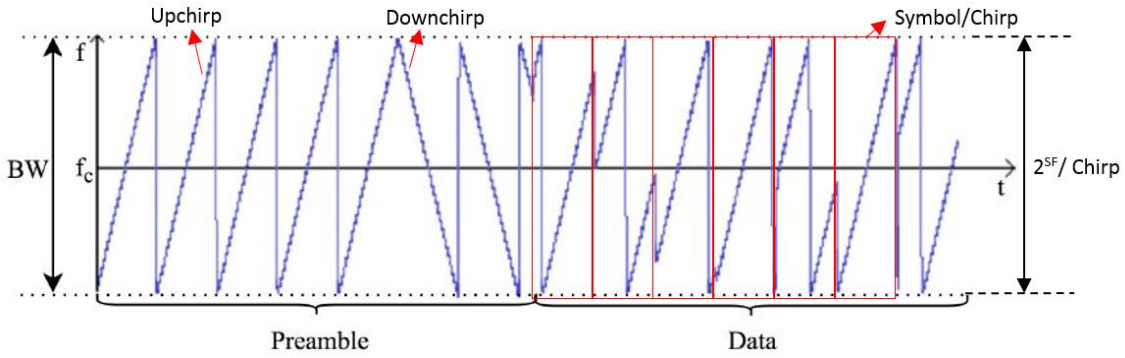


Figura 3.2: Exemplo de uma transmissão LoRa através da variação da frequência no tempo. Adaptado de [49].

3.1.2 Pacote LoRa

A Figura 3.3 representa a estrutura de um pacote LoRa. Um pacote LoRa é constituído por dois componentes essenciais, o *preamble* e a *payload*. Cada trama é iniciada com o *preamble* e este tem como objetivo sincronizar o transmissor e o recetor. O *preamble* pode ser programado e o tamanho deste pode variar entre 10.25 e 65539.25 *symbols*. Este é constituído por uma componente obrigatória de 4.25 *symbols* e por uma componente programável mínima de 6 *symbols*. Por predefinição este apresenta um tamanho de 12.25 *symbols*. Opcionalmente, a seguir ao *preamble* pode ser colocado o *header*. Quando este está presente denomina-se de pacote LoRa explícito, caso não esteja designa-se de pacote LoRa implícito. O *header* contêm informações acerca do tamanho da *payload* (em *bytes*), o CR a utilizar para o resto da comunicação e indica se no final da transmissão exsite ou não um *Cyclic Redundancy Check* (CRC) de 16 bits. Uma vez definidos este

parâmetros estes irão ser mantidos, tanto no lado do transmissor como no do recetor, até que sejam alterados. O tamanho da *payload* é armazenado num único *byte* limitando assim o tamanho deste para um valor máximo de 255 *bytes*. O *header* inclui também um CRC para permitir que o recetor descarte pacotes inválidos. Além disso, quando o *header* está presente este é sempre transmitido com um CR de 4. O *header* é opcional para permitir que possa ser desativado em situações onde não seja necessário, por exemplo quando as informações presentes no *header* já são conhecidas e não se pretendam alterar, resultando numa diminuição do tempo de transmissão. A *payload* é enviada a seguir ao *header* e esta é codificada com um CR variável definido no *header*. A *payload* ou contém pacotes de dados ou pacotes de controlo referentes à camada MAC do LoRaWAN. Opcionalmente, a *payload* pode ser seguida de um *payload* CRC caso tenha sido definido no *header*. Um esquema resumo da estrutura de um pacote LoRa encontra-se representado na Figura 3.3 [53][56].

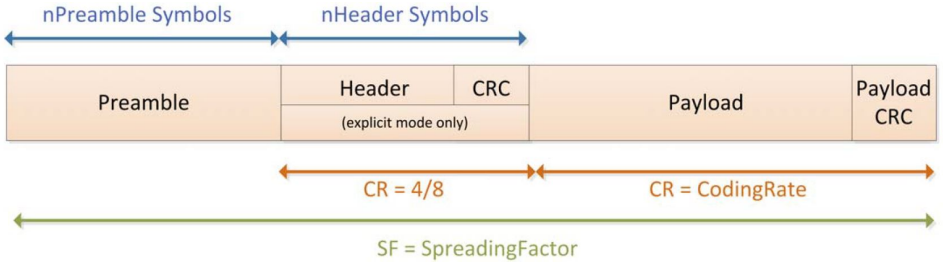
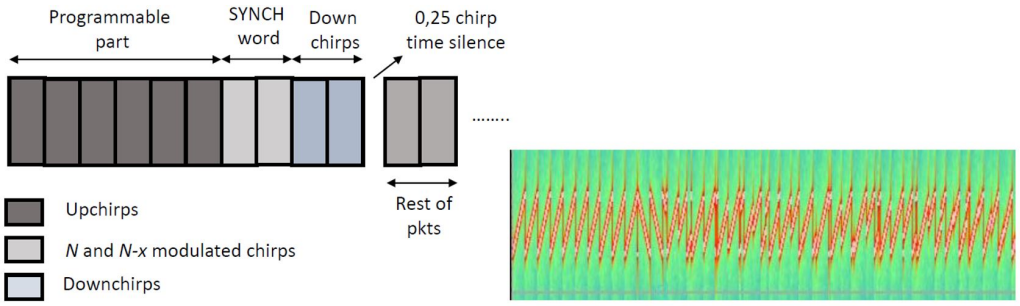


Figura 3.3: Estrutura de um pacote LoRa [56].

Uma vez que a modelação LoRa é patenteada os detalhes do seu funcionamento nunca foram divulgados publicamente no entanto, alguns investigadores descobriram alguns pormenores acerca do seu funcionamento. Segundo Seller e Sorin [57], o *preamble* inicia sempre com uma sequência de *upchirps*, que são programaveis e ajudam a detetar o início de uma trama. A parte programável é seguida de dois *chirps* que contêm um parâmetro de sincronização denominado de *synch word*. Este parâmetro permite também distinguir dispositivos que operam em redes diferentes através da utilização de um valor diferente para cada rede. De seguida, o *preamble* termina com dois *downchirps* também utilizados para sincronizar a frequência. Após isto, o transmissor permanece em silencio durante o tempo necessário para enviar um quarto de um *symbol* (0.25 *symbol*). Este tempo é utilizado pelo recetor para efetuar ajustes à frequência de modo a obter a melhor sincronização possível. A estrutura do *preamble* encontra-se representado na Figura 3.4.a, enquanto que a Figura 3.4.b representa um espectrograma de um pacote LoRa. Através deste espectrograma é possível observar facilmente a sequência de *upchirps* presentes no início do *preamble* [57].



(a) Estrutura do *Preamble* (b) Espectograma de um pacote LoRa

Figura 3.4: *Preamble* de um pacote LoRa [57].

3.2 LoRaWAN

LoRaWAN define o protocolo de controlo de acesso ao meio (MAC). Este é um protocolo *open source* desenvolvido pela LoRa Alliance e que opera sobre a camada física LoRa. O protocolo LoRaWAN fornece os mecanismo de controlo de acesso ao meio necessários para permitir comunicações entre múltiplos dispositivos e a *gateway(s)* da rede. Este protocolo desempenha um papel importante, uma vez que influencia diretamente o desempenho diversos parâmetros relacionados com rede e os seus diferentes constituintes, como por exemplo a duração da bateria do dispositivo, a capacidade da rede, a qualidade do serviço e a segurança da própria rede [57].

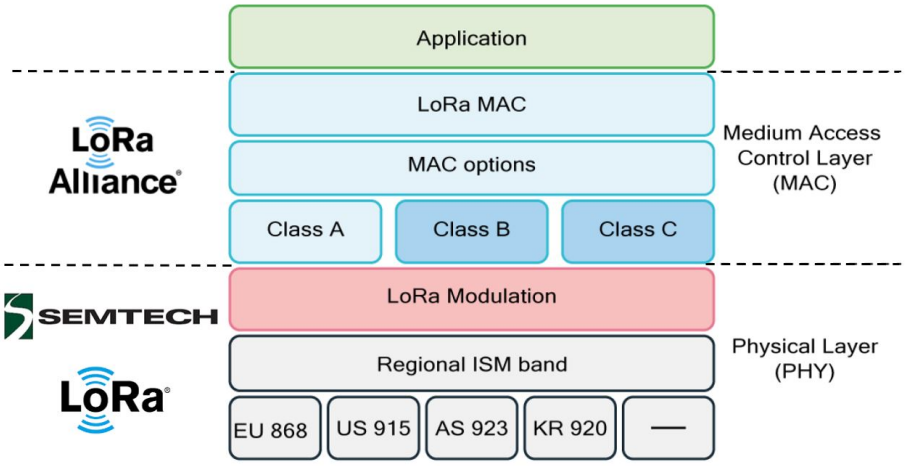


Figura 3.5: Pilha protocolar da comunicação LoRa/LoRaWAN. Adaptado de [55].

3.2.1 Componentes de uma Rede LoRaWAN

Um dos objetivos principais do LoRaWAN é maximizar o tempo de vida da bateria dos dispositivos e, de modo a optimizar este parâmetro, a topologia da rede tipicamente utilizada é em estrela. Muitas tecnologias utilizam topologia em malha de modo a aumentarem o alcance, como é o caso do Zigbee que já foi referenciado anteriormente na secção 2.3.1, mas em contrapartida diminui a capacidade da rede, aumenta a complexidade da mesma e diminui a duração da bateria, dado que os dispositivos irão receber e reenviar informação de outros dispositivos que provavelmente será irrelevante para os mesmos. A Figura 3.6 representa a topologia de uma rede LoRaWAN.

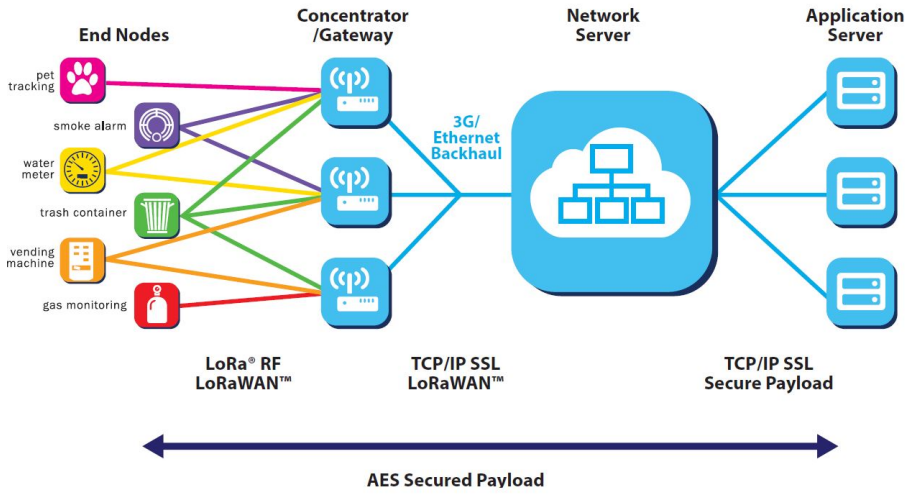


Figura 3.6: Topologia de uma rede com o protocolo LoRaWAN [58].

Um rede LoRaWAN é possui três componentes essenciais: *end node*, *gateway* e *network server*.

- *End node*: É tipicamente um dispositivo de baixo custo e de baixo consumo energético, equipado com sensores ou atuadores e que comunica com a *gateway* utilizando a comunicação LoRa.
- *Gateway*: É um dispostivo intermédio de tem como função reencaminhar pacotes provenientes dos *end nodes* para o *network server* e vice-versa. Este funciona como um interface que do lado dos end nodes opera com uma comunicação LoRa e do lado do *network server* opera com uma comunicação IP que permite uma maior transferência de dados.
- *Network server*: Responsável pelo processamento de dados e outras tarefas tais como deteção de pacotes duplicados, descodificação dos pacotes, entre outras.

Uma particularidade presente numa rede LoRa/LoRaWAN é o facto de os *end nodes* não estarem associados a uma *gateway* específica. Tipicamente, os dados transmitidos são recebidos por várias *gateways* que posteriormente irão reenviar os dados para o servidor da rede ou para a *cloud* (representado como *Network Server* na Figura 3.6) através de uma rede IP secundária. Por sua vez, o servidor da rede será responsável pela gestão da rede, necessitando de capacidades computacionais para exercer diversas tarefas tais como a deteção de pacotes de dados duplicados, a escolha da melhor *gateway* para envio da confirmação dos dados recebidos e a realização de rotinas de segurança. Esta é uma caraterística fundamental para aplicações que necessitem do rastreamento de bens/recursos, uma vez que a mobilidade destes pode implicar a necessidade de conexão a diferentes *gateways* consoante a sua localização [50][59].

Outra característica dos dispositivos de uma rede LoRaWAN é que estes são assíncronos o que permite aumentar a longevidade da bateria, uma vez que não é necessário uma constante sincronização entre os dispositivos. Além disso os *end nodes* apenas comunicam quando possuem dados para enviar. O envio tanto pode ser periódico, como pode ser gerado por um evento em particular [49][50].

O LoRa, assim como outras tecnologias de banda estreita, necessita de ser eficiente uma vez que qualquer sobrecarga provocará um aumento da latência e do consumo energético. De forma a superar este problema, LoRaWAN utiliza uma estratégia simples para a gestão do meio de transmissão de forma a manter a complexidade da rede o mais simples possível e simultâneamente diminuindo os custos energéticos. Para isso, LoRaWAN adotou o protolo Aloha puro, mas com a adição de mecanismos de acknolegement (ACK) de modo a simplificar o acesso ao meio [53].

É importante referir que o LoRaWAN não permite a comunicação direta entre *end nodes*. Os pacotes de dados apenas podem ser enviados de um *end node* para um *network server* e vice-versa [54].

As bandas de frequência ISM estão sujeitas a regulamentações no que diz respeito à potência máxima de transmissão e do *duty cycle*. As limitações do *duty cycle* traduzem-se na introdução de tempos de espera entre o envio de diferentes mensagens. Uma vez que as redes LoRaWAN operam nestas bandas de frequência, estes regulamentos devem ser tidos em consideração [54].

3.2.2 Classes - LoRaWAN

Conforme as necessidades de cada um dos dispositivos o protoloco LoRaWAN permite definir diferentes classes, classe A, B e C. Na figura 3.7 encontra-se representado um gráfico com as diferentes classes e demonstra a relação entre a latência de comunicação e duração da bateria. Estas classes diferenciam-se entre si essencialmente pela latência de comunicação das mensagens de *downlink*

(mensagens enviadas do servidor para os dispositivos, como por exemplo uma mensagem de *acknowlegement*). Porém, quanto menor for esta latência menor será a duração da bateria. A partir da Figura 3.7 é possível observar que os equipamentos que operem na classe A terão uma latência de comunicação superior às restantes classes no entanto, a bateria terá uma duração superior. O oposto acontece no dispositivos que utilizem a classe C, estes possuem uma latência de comunicação inferior às restantes classes contudo, o tempo de vida da bateria também será inferior. Por sua vez, a classe B encontra-se numa situação intermédia em relação às classes A e C.

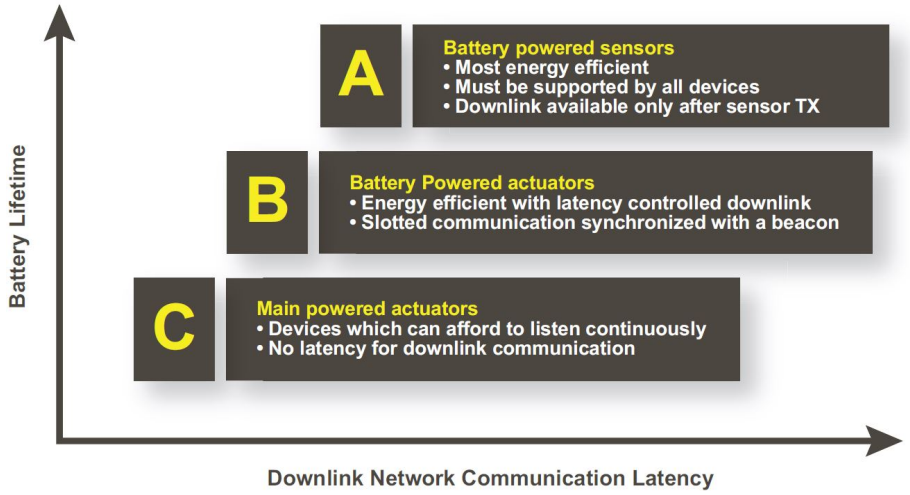


Figura 3.7: Diferentes classes para os dispositivos do protocolo LoRaWAN [50].

Todas as classes suportam comunicações bidirecionais e todos os dipositivos que utilizam o protocolo LoRaWAN devem possuir pelo menos as caraterísticas básicas, isto é, as caraterísticas de um dispositivo de classe A. Classe A e B são tipicamente utilizadas em dispositivos que requerem uma bateria como fonte de energia, enquanto que os de classe C são aplicados em dispositivos que possuem uma fonte de energia constante, onde conservar energia não é fundamental como no caso das classes A e B.

Como já foi referido anteriormente as diferentes classes diferenciam-se entre si pela latência de comunicação de mensagens de *downlink*. Os dispositivos que utilizam a classe A apenas disponibilizam duas janelas de curta duração para a receção de mensagens após uma transmissão. Posteriormente o dispositivo entra em modo *sleep* para preservar a bateria. Caso o servidor não consiga comunicar nestes dois períodos, este terá de esperar por uma nova transmissão do dispositivo e pelas respetivas janelas para o poder fazer. Esta classe é tipicamente utilizada para dispositivos alimentados a bateria e que têm como objetivo

efetuar leituras de sensores. A classe B tem por base o funcionamento da classe A, porém esta permite, adicionalmente, a abertura de janelas de receção extras em tempos calendarizados. Estas janelas são sincronizadas através de mensagens enviadas pela *gateway* denominadas de *beacons*. Assim como na classe A, a classe B é geralmente utilizada em dispositivos alimentados a bateria, porém a sua funcionalidade é tipicamente para o controlo de atuadores. Na classe C, o dispositivo encontra-se constantemente com a janela de receção aberta com a exceção de quando o próprio dispositivo se encontra a transmitir dados. Isto permite uma diminuição significativa de latência na receção de mensagens, no entanto implica um aumento substancial do consumo energético comparativamente às outras classes. Por esta razão, este tipo de classe é tipicamente aplicado em dispositivos que possuem uma fonte de energia constante como já foi referido anteriormente. A Figura 3.8 representa o modo de operação das diferentes classes do protocolo LoRaWAN [59][58].

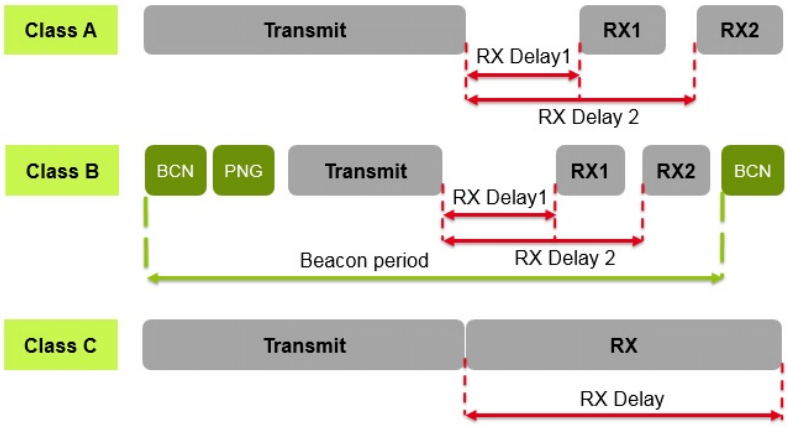


Figura 3.8: Modo de operação das diferentes classes do protocolo LoRaWAN [60].

3.2.3 Pacotes MAC - LoRaWAN

LoRaWAN tem por base o pacote LoRa descrito no capítulo 3.1.2. Como foi referido neste, o *payload* pode conter pacotes de dados ou pacotes de dados referentes à camada MAC do LoRaWAN, sendo o objeto de estudo deste capítulo. O formato completo de um pacote MAC encontra-se representado na Figura 3.9 porém, devido à complexidade do mesmo, apenas será realizado uma análise geral.

O *payload*, representado na Figura 3.9 por PHYPayload, encontra-se dividido em três partes, o MAC header (MHDR), o MAC Payload e o MIC.

O MAC header (MHDR) especifica o tipo de mensagem MAC e a versão do formato da mensagens.

O MAC Payload é constiuído por um *frame header* (FHDR) e por dois outros *frames* opcionais com informações adicionais acerca da mensagem MAC. O FHDR possui o endereço do *end node* (DevAddr) e um conjunto de parâmetros de controlo utilizados no envio de comandos MAC.

O MIC significa (*message integrity code*) e é uma mensagem que tem como objetivo proteger a integridade dos dados, nomeadamente o MHDR e o MAC Payload [54][59].

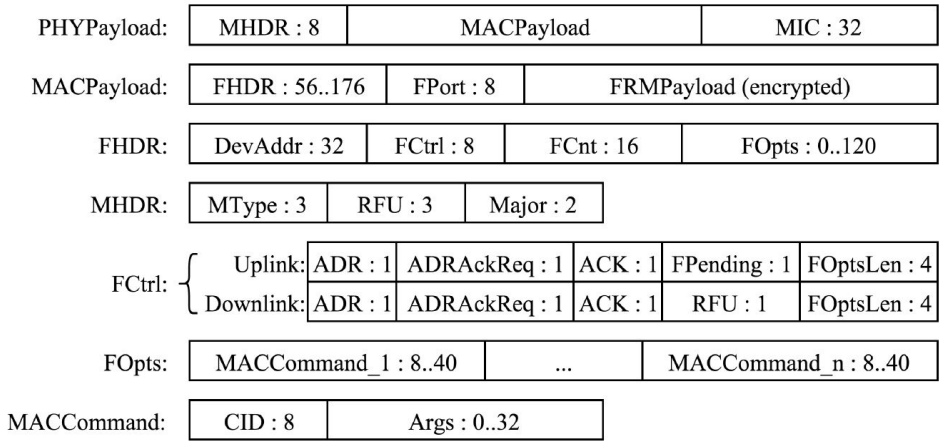


Figura 3.9: Estrutura de um pacote LoRaWAN. O tamanho dos campos estão em *bits* [54].

3.2.4 Segurança

Em termos de segurança o protocolo LoRaWAN utiliza duas camadas de segurança, uma para a rede e uma para a aplicação. A camada de segurança da rede garante a autenticidade do dispositivo na rede enquanto que a camada para aplicação garante que operador da rede não tem acesso aos dados da aplicação do utilizador [54][61]. Toda a informação encontra-se protegida com encriptação AES (Advanced Encryption Standard) de 128 *bits* constituída por 4 componentes:

- **DevAddr:** O identificador do dispositivo de 32 bit.
- **AppEUI:** O identificador global único da aplicação, IEEE EUI64.
- **NwkSKey:** Uma *Network session key* (chave AES-128) usada para encriptar a comunicação entre o *end node* e o *network server*.

- **AppSKey:** Uma *Application session key* (chave AES-128) utilizada para proteger a informação específica da aplicação.

3.3 Aplicações da Tecnologia LoRa/LoRaWAN

O objetivo deste subcapítulo será abordar algumas aplicações já implementadas que utilizam a tecnologia LoRa/LoRaWAN como protocolo de comunicação. Devido às características que o LoRA/LoRaWAN proporciona, esta tecnologia é tipicamente utilizada em implementações de grande escala e para aplicações pouco sensíveis à latência. As áreas de aplicação mais comuns são as cidades inteligentes, a agricultura inteligente, a monitorização ambiental e área da saúde, no entanto não se encontra limitado a estas podendo ser aplicada a outras áreas da IoT.

Em relação à área das cidades inteligentes existem atualmente aplicações relacionadas com os transportes públicos. No México, a empresa Ursalink disponibiliza soluções de rastreamento de autocarros que tem por base a tecnologia LoRa em vez dos tradicionais sistemas de comunicação móveis, que são mais dispendiosos. Além disso, o servidor utilizado para a aplicação permite estimar o tempo de chegada de um autocarro à proxima paragem e envia o resultado para esta [62]. Semelhante a esta solução, em Montreal, no Canadá, a X-Telia em parceria com a Semtech desenvolveram horários para autocarros inteligentes. Estes consistem em monitores localizados nos respetivos pontos de paragem, alimentados por um painel solar e que recebem informações através de um sistema de comunicações LoRa acerca dos horários. Em caso de algum atraso os monitores são atualizados em tempo real com o respetivo tempo [63].

Chipsafer foi uma das primeiras empresas a utilizar a tecnologia LoRa no segmento da agricultura inteligente. Esta fornece um serviço de gestão de gado através da colocação de um dispositivo externo em cada animal. Cada dispositivo permite obter informações em tempo real acerca da sua localização e respetiva movimentação, assim como a temperatura e a humidade do local onde este se encontra. Toda a informação é armazenada na *cloud* para que possa ser analisada e caso seja detetada alguma anomalia (como por exemplo caso um animal se encontre fora de uma área específica) esta despertará um alerta [64][65]. Quantified AG é outra empresa que fornece um serviço de gestão de gado semelhante ao da Chipsafer, no entanto esta tem como objetivo principal monitorizar a saúde dos animais. Através de etiquetas eletrónicas colocadas nas orelhas dos animais é possível não só obter diversas informações acerca da localização e movientação dos animais, mas também permite medir a temperatura e outros sinais vitais, de modo a detetar antecipadamente problemas de saúde com os animais e desta forma efetuar o tratamento o mais cedo possível e aumentar a probabilidade de recuperação. Assim como a solução da Chipsafer, a Quantified AG

amazena e processa toda a informação na *cloud*. Em caso de anomalias este gera notificações para diversos equipamentos tais como smartphones ou tables através de um email ou SMS. Toda a informação pode ainda ser consultada através de uma página *web*. [66][67].

A tecnologia LoRa já se encontra atualmente em soluções na à área da monitorização ambiental. Em 2015 na cidade de Lyon, na França, a empresa Birdz implementou um sistema de gestão de água com o objetivo de aumentar a eficiencia de utilização deste recurso. Este sistema permite não só monitorizar o consumo de água mas também detetar falhas no sistema (como por exemplo ruturas nos sistemas que originam o vazamento de água) e que consequentemente levam a uma redução do desperdício deste recurso. Desde 2015 até 2018, este sistema constituído por cerca de 400 mil dispositivos equipados com a tecnologia LoRa, permitiu detetar cerca de 1200 ruturas no sistema de distribuição de água da cidade, reduzindo o desperdício de água em cerca de 1 milhão de metros cúbicos de água anualmente, representando um aumento de 8 % na eficiência do sistema de distribuição [68][69]. Ainda neste segmento da IoT, a Green Stream é uma empresa que dispõe soluções para a deteção de inundações. Devido ao baixo consumo energético da tecnologia LoRa estes sistemas autónomos não necessitam de alimentação externa. Por usa vez, estes são alimentados por um painel solar. Além disto, encontram-se equipados com sensores ultrassônicos que permitem indicar não só as ruas que se encontram inundadas mas também a profundidade das mesmas e o estado atual da situação, isto é, se a profundidade está a aumentar ou diminuir. O objetivo desta solução é reduzir o custo de bens materiais e de vidas provocadas pelas inundações [70].

No que diz respeito à área de saúde existem aplicações que utilizam o sistema de comunicação LoRa no entanto grande parte destas representam soluções de rastreamento. Lineable e Careband são duas empresas que desenvolveram soluções para pessoas que sofram de Alzheimer ou Demência. As pessoas que sofrem deste tipo de problema por vezes perdem o sentido de orientação, acabando por não conseguir retornar a casa. Estas empresas desenvolveram sistemas vestíveis de rastreamento em tempo real para pacientes que sofram destes problemas com o objetivo de localizar mais rapidamente estes indivíduos de modo a garantir a sua segurança e simultâneamente proporcionar tranquilidade aos familiares e cuidadores [71][72]. Apesar de atualmente grande parte das aplicações na área da saúde envolverem sistemas de rastreamento, a Semtech disponibiliza soluções para outros problemas dentro deste segmento da IoT. Uma delas é um sistema de deteção de quedas desenvolvido especialmente para idosos. O objetivo desta solução é reduzir o tempo que um idoso permanece no chão, uma vez que quanto maior for a duração maior a probabilidade de desenvolver outros problemas de saúde com consequências graves. O principio de funcionamento é bastante simples, um idoso encontra-se equipado com um

sensor embebido com um módulo de rádio LoRa. Estes são responsáveis por enviar dados para a *cloud* onde serão analisados e, caso seja detetada alguma anomalia, isto é, seja detetada uma queda, será gerado um alerta indicando o local onde ocorreu a queda para que os cuidados sejam efetuados o mais rapidamente possível [73]. Outra solução que a Semtech fornece é um sistema de monitorização de frigoríficos médicos. Estes encontram-se equipados com sensores que permitem medir o consumo de energia e a temperatura. A informação recolhida é posteriormente enviada para a *cloud* para ser analisada. Caso seja detetada uma falha de energia, o sistema é capaz de calcular o ritmo com que a temperatura desce no interior do frigorífico, o tempo que este se encontrou sem energia e se durante a falha a temperatura se manteve adequada aos conteúdos presentes neste [74].

3.4 Equipamentos LoRa

Atualmente existe uma grande variedade de equipamentos que utilizam a tecnologia LoRa. Apesar de a LoRa ser propriedade da Semtech, esta forneceu uma licença a outras empresas, nomeadamente a Microship, STMeletronics e HopeRF de forma a estas poderem desenvolver produtos baseados nesta tecnologia. Nesta secção será abordado alguns dos diferentes equipamentos existentes no mercado, com principal incidência nos da Semtech. É importante salientar a existência de uma organização sem fins lucrativos denominada de LoRa Alliance, que tem como objetivo oferecer suporte e promover a adoção do *standard* LoRa e LoRaWAN de forma a assegurar a interoperabilidade entre todos os produtos e tecnologias. Além disso, esta organização dispõe de um processo de certificação de equipamentos de modo a garantir ao utilizador que estes são fiáveis e seguem as normas corretas do protocolo LoRaWAN [75].

Os dispositivos LoRa podem-se dividir em duas categorias diferentes, nós LoRa (também designados de *end nodes*) e LoRa *gateways*, apresentando cada uma destas algumas diferenças em termos de constituição [76].

3.4.1 LoRa *Gateways*

A Figura 3.10 apresenta a constituição básica de uma LoRa *gateway*. Esta é tipicamente constituída por dois componentes essenciais, um processador ou microcontrolador e um ou vários módulos de rádio. A comunicação entre o módulo de rádio e a unidade de processamento é efetuada através de um interface SPI (*Serial Peripheral Interface*). O processador possui diversas funções, tais como efetuar uma leitura dos diversos canais e se necessário realizar a desmodulação dos sinais provenientes dos módulos de rádio.

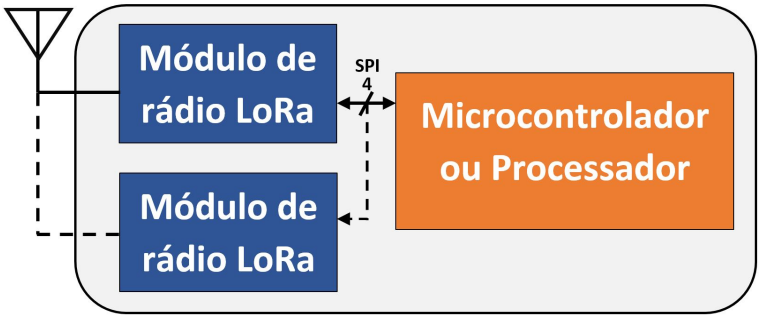


Figura 3.10: Constituição básica de uma LoRa *gateway*.

Os processadores disponibilizados pela Semtech normalmente utilizados numa LoRa *gateway* são o SX1301, SX1302 e o SX1308. Estes são bastantes semelhantes entre si, no entanto diferenciam-se pelo seu ambiente de aplicação, isto é, o SX1301 e o SX1302 foram desenvolvidos para operar em ambientes externos enquanto que o SX1308 foi desenvolvido para operar em ambientes internos. Além disso, estes processadores possuem 10 canais de receção programáveis e conseguem suportar até 10 000 nós. Relativamente aos módulos de rádio, o SX1255, SX1257 e SX1258 são os mais comuns. Estes são semelhantes apenas variando entre si a banda de frequência em que operam. Na Tabela 3.2 encontra-se resumida os diferentes componentes abordados acerca da LoRa *gateway* [76].

Tabela 3.2: Componentes tipicamente utilizados numa LoRa *gateway* [76].

Chip	**Frequência de Op. (MHz)**	**Sensibilidade máxima na receção (dBm)**	**Observações**
SX1255	400-510		
SX1257	860-1000		
SX1258	779-787		Desenvolvido especialmente para o mercado chinês.
SX1301		-142.5 (SX1257) -137.5 (SX1255)	Desenvolvido para o uso exterior. Temp. de operação: -40 ºC até 85 ºC
SX1302		-141 (SX1255, SX1257)	Segunda geração do SX1301. Mais barato e com maior eficiência energética.
SX1308		-139 (SX1255, SX1257)	Desenvolvido para o uso interior. Temp. de operação: 0 ºC até 70 ºC

3.4.2 LoRa *End Nodes*

A constituição de um nó LoRa é mais simples do que uma *gateway*. Os nós são geralmente constituídos por um microcontrolador e um módulo de rádio. Dependendo do objetivo do nó diversos sensores podem ser associados a este. Assim como nas *gateways*, a comunicação entre o módulo de rádio LoRa e o microcontrolador é realizado através de um interface SPI. A Figura 3.11 representa um esquema da constituição básica de um nó LoRa.

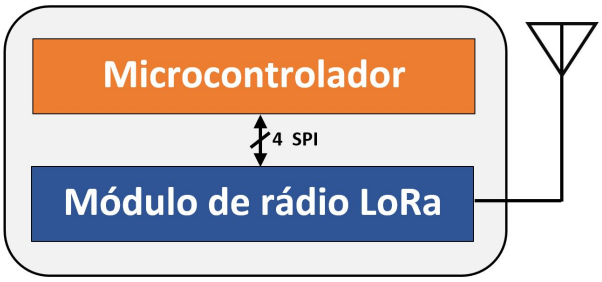


Figura 3.11: Constituição básica de um nó LoRa.

O microcontrolador utilizado nos nós LoRa tem como objetivo principal funcionar como um meio de ligação entre um sensor e o módulo de rádio. Não existem microcontroladores específicos para nós LoRa, sendo que a escolha deste componente é realizado segundo os requisitos exigidos pela aplicação sobre a qual se pretende implementar o nó. Em relação aos módulos de rádio estes diferem dos módulos já abordados para as *gateways*. Na Tabela 3.3 encontram-se os módulos de rádio existentes no mercado produzidos pela Semtech, bem como algumas das suas características [76].

Tabela 3.3: Módulos de rádio produzidos pela Semtech para nós LoRa [76].

Chip	Frequência de Op. (MHz)	Potência do sinal emitido (dBm)	Observações
SX1261	150-960	15	Suportam todas as principais bandas de rádio ISM inferiores a 1 GHz
SX1262	150-960	22	
SX1268	410-810	22	Desenvolvido especialmente para o mercado chinês.
SX1272 / SX1273	860-1020	20	
SX1276 / SX1277	137-1020	20	Suportam todas as principais bandas de rádio ISM inferiores a 1 GHz
SX1278	137-525	20	
SX1279	137-960	20	Suportam todas as principais bandas de rádio ISM inferiores a 1 GHz

Como já foi referido no início desta secção, apesar que a tecnologia LoRa ser propriedade da Semtech esta forneceu licenças a outras empresas para a produção de módulos de rádio baseados nesta tecnologia. A Tabela 3.4 apresenta alguns módulos de rádio dos fabricantes HopeRF e Microchip, assim como algumas das suas caraterísticas [77][78].

Tabela 3.4: Módulos de rádio produzidos pela Semtech para nós LoRa [77][78].

Chip	Fabricante	Frequência de Op. (MHz)	Potência do sinal emitido (dBm)	Baseado no	Marca no Chip
RFM95W	HopeRF	868-915	20	SX1276	RF96
RFM96W	HopeRF	433-470	20	SX1276	RF96
RFM98W	HopeRF	433-470	20	SX1278	RF98
RN2483A	Microchip	433/868	10/14	SX1276/78	-
RN2903	Microchip	915	18.5	SX1276	-

Atualmente existe uma enorme variedade de nós LoRa disponíveis no mercado. De modo a simplificar a análise destes equipamentos estes vão ser divididos em três grupos distintos, nomeadamente em placas de desenvolvimento, LoRa *radio shields* e os próprios módulos de rádio.

As placas de desenvolvimento LoRa caraterizam-se por integrarem um módulo de rádio, um microcontrolador e uma antena numa placa de circuito im-

presso (PCB). Estes são os componentes básicos deste tipo de placas, no entanto estas podem possuir outros componentes extra consoante as necessidades, como por exemplo um módulo GPS, um LCD, outros módulos de comunicação, entre outras. A grande vantagem das placas de desenvolvimento são a sua versatilidade no desenvolvimento de uma aplicação devido à grande quantidade de funcionalidades que estas possuem porém, isto provoca um aumento do custo e das dimensões do equipamento. No entanto, na maioria das situações, após o desenvolvimento da aplicação a totalidade das funcionalidades não são utilizadas resultando num aumento de custo e de dimensões desnecessário.

A Wemos TTGO LoRa32 apresentado na Figura 3.12 é um exemplo de uma placa de desenvolvimento para um nó LoRa. Esta encontra-se equipada com o módulo de rádio SX1276 que permite uma comunicação LoRa a uma frequência de 868 MHz. Além disso possui uma alta sensibilidade, superior a 148 dBm e uma potência de saída de 20 dBm possibilitando comunicações até 2 km. Adicionalmente, este é um *System on a Chip* (SoC) ESP32, integrando num só *chip* um microcontrolador e os sistemas de comunicações LoRa, Wi-Fi e Bluetooth. Por último, este inclui ainda um ecrã oLED de 0,96 polegadas [79].

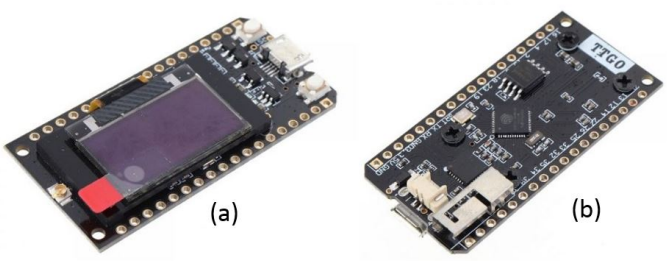


Figura 3.12: Wemos TTGO LoRa32: (a) face superior, (b) face inferior [79].

Outra placa de desenvolvimento é a LoPy4 produzida pela Pycom, sendo esta desenvolvida essencialmente para ser aplicada a um nível empresarial. Assim como Wemos TTGO LoRa32 este é um produto baseado num SoC ESP32 incluindo os sistemas de comunicação LoRa, Wi-Fi e Bluetooth e o microcontrolador num só *chip*, podendo este ser programado em *MicroPython*. Para comunicação LoRa este utiliza o módulo de rádio SX1276 permitindo comunicações até 40 km em modo nó LoRa. Porém, este tem a capacidade de operar como uma *nano gateway* com a capacidade para suportar apenas 100 nós LoRa no entanto, neste modo de funcionamento, o alcance máximo é reduzido para 22 km. Ainda em relação às comunicações este inclui ainda um módulo de rádio SigFox. Em relação ao consumo energético, este possibilita consumos significativamente inferiores em relação a equipamentos semelhantes [80].

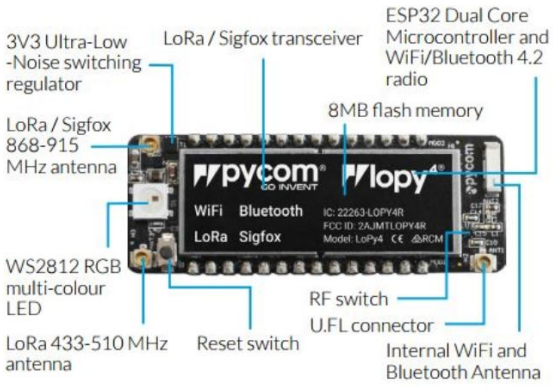


Figura 3.13: LoPy4 [80].

Os LoRa *radio shields* são semelhantes às placas de desenvolvimento no entanto diferenciam-se destas pela inexistência de um microcontrolador integrado. Estas possuem dois componentes essenciais integrados numa PCB, um módulo de rádio LoRa e uma antena.Adicionalmente, estes podem possuir outros periféricos como por exemplo um módulo GPS. Os LoRa radio shields foram desenvolvidos para serem facilmente acopolados a placas de desenvolvimento mais comuns como é o caso do Arduíno, STM32 Nucleo e do Raspberry Pi. Devido à falta de um microcontrolador, na generalidade dos casos os LoRa radio shields possuem um custo inferior quando comparadas às placas de desenvolvimento.

O Dragino LoRa shield foi desenvolvido para ser emparelhado com as placas de desenvolvimento Arduino Leonardo, Uno, Mega, DUE. Esta encontra-se disponível com dois módulos de rádio LoRa distintos, o SX1276 e o SX1278, suportando todas as principais bandas de rádio ISM inferiores a 1 GHz [81].

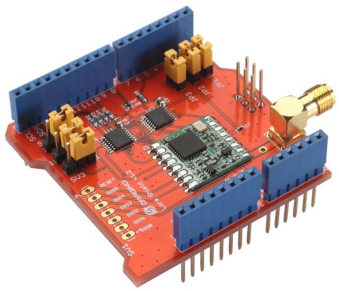


Figura 3.14: Arduino LoRa shield [81].

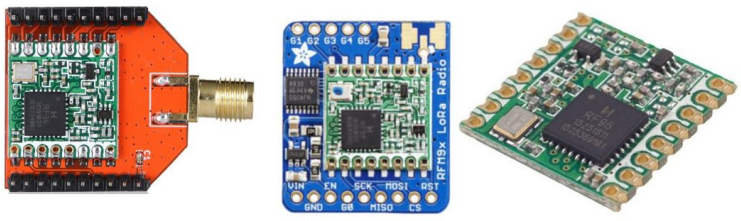
O SX1276MB1MAS apresentado na Figura 3.15 é um de vários LoRa shields especialmente desenhado para placas de desenvolvimento da família STM32 Nucleo. Este possui um módulo de rádio SX1276 e dois conectores para an-

tenas que permitem que este opere na gama de frequências dos 433 e 868 MHz [82].

Figura 3.15: Mbed LoRa Shield [82].

Por último, existem os próprios módulos de rádio que são os equipamentos mais simples e com um menor custo quando comparadas com as placas de desenvolvimento e os LoRa *radio shields*. Além disso são mais compactos e possibilitam uma maior liberdade na escolha dos restantes componentes necessários para a aplicação desejada. Por outro lado, todas as ligações entre os componentes tem ainda de ser realizada e, caso seja necessário, o desenvolvimento de uma PCB. Estas tarefas, por vezes, podem ser morosas, sendo esta uma desvantagem da utilização destes equipamentos.

Na Figura 3.16 encontra-se representado dois módulos de rádio, o Dragino LoRa Bee e o Adafruit RFM95W. Ambos possuem o módulo de rádio RFM95W produzido pela empresa HopeRF. Este é baseado no módulo SX1276 e que opera gama de frequências de 868 e 915 MHz. Estes módulos são por vezes colocados em placas adaptadoras para *breadboard* para facilitarem a sua utilização [83][84][85].



(a) (b) (c)

Figura 3.16: Módulos de rádio LoRa: (a) Dragino LoRa Bee [83], (b) Adafruit RFM95W [84], (c) RFM95W [85].

3.5 Módulo de Rádio LoRa - SX1276

O objetivo deste capítulo é realizar uma análise geral ao *datasheet* do módulo de rádio LoRa SX1276. Pretende-se salientar os aspetos mais importantes presentes no *datasheet* e simultaneamente entender o funcionamento do módulo de rádio. Todo o conteúdo presente neste capítulo terá por base o *datasheet* [56].

3.5.1 Caraterísticas Gerais

O *datasheet* apresentado serve de base não só para módulo de rádio SX1276, mas também para o SX1277/78/79, uma vez que na sua generalidade são semelhantes apresentando no entanto algumas diferenças. As diferenças mais importantes referem-se às bandas de frequência e aos valores do SF disponíveis. Os módulos SX1276, SX1277 e SX1279 permitem operar em todas a bandas de frequência ISM disponíveis porém, o módulo SX1278 apenas opera nas frequências mais baixas. Em relação ao SF todos os modelos permitem selecionar valores de SF entre 6 a 12 com a exceção do módulo SX1277 que apenas permite escolher de 6 a 9. A Tabela 3.5 resume os parâmetros chave e as variações entre os diferentes módulos.

Tabela 3.5: Módulos de rádio LoRa SX1276/77/78/79 [56].

Modelo	Frequência (MHz)	SF	BW (MHz)	Bit Rate (Kbps)
SX1276	137 - 1020	6 - 12	7.8 - 500	0.18 - 37.5
SX1277	137 - 1020	6 - 9	7.8 - 500	0.11 - 37.5
SX1278	137 - 525	6 - 12	7.8 - 500	0.18 - 37.5
SX1279	137 - 960	6 - 12	7.8 - 500	0.18 - 37.5

Todos os módulos são transceptores bidireccionais, porém *half-duplex*. O acesso a todos os registos de configuração acerca de parâmetros relativos à modelação/desmodelação rádio, assim como outros aspetos relativos a parâmetros digitais podem ser configurados e acedidos através de uma interface SPI. Além disso todos os módulos de rádio encontram-se equipados com um *modem* LoRa e com um *modem* para o *standard* FSK, permitindo que este módulo opere com base nestas duas tecnologias, no entanto nunca em simultâneo. Visto que a tecnologia LoRa é o objetivo deste trabalho apenas esta será abordada.

A tensão de alimentação pode variar entre os 1.8 V e os 3.7 V, no entanto a tensão ótima de alimentação deve ser de 3.3 V. A faixa de temperatura que estes dispositivos devem operar é entre os -40 ºC e os 85 ºC. Por predefinição, a modulação LoRa terá os parâmetros referidos a Tabela 3.6.

Tabela 3.6: Parâmetros predefinidos no módulo LoRa [56].

Parâmetro	Unidade
Frequência do oscilador (f_{xos})	32 MHz
Bandwidth (BW)	125 kHz
Spreading Factor (SF)	12
Coding Rate (CR)	4/6
CRC no payload	Ativado
Tamanho do payload	64 bytes
Tamanho do preamble	12 symbols
Potência de transmissão	13 dBm
Packet Error Rate (PER)	1 %

Na Figura 3.17 é apresentado um diagrama de blocos simplificado do módulo SX1276. Neste capítulo apenas será efetuada uma visão geral acerca deste, porém alguns dos seus constituintes serão analisados com maior detalhe nos capítulos seguintes.

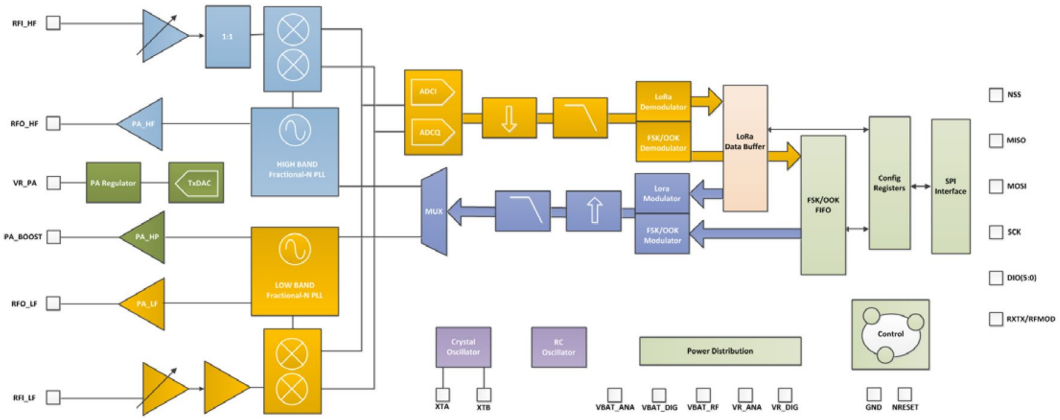


Figura 3.17: Diagrama esquemático de blocos do módulo SX1276 [56].

O lado esquerdo do diagrama representa os diferentes componentes presentes na fase inicial do processo de receção e a fase final do processo de transmissão de informação. No que diz respeito à fase inicial de receção de dados, este é constituído por dois *low noise amplifiers* (LNA) e dois *mixers*. De seguida é utilizado um par de sigma-delta ADCs que permitem converter o sinal recebido para o domínio digital sendo posteriormente realizada a desmodulação e todo processamento de sinal necessário. Em relação à fase final de transmissão, este é constituído por um *multiplexer*, seguido de um filtro passa alto e passa baixo e três amplificadores. Dois destes, designados no diagrama de `PA_LF` e `PA_HF`, permitem potências fixas de transmissão de +14 dBm e podem conectados dire-

tamente aos respetivos recetores possibilitando assim obter um transceptor de alta eficiência. O terceiro amplificador, designado de `PA_HP` permite potências de transmissão de +20 dBm. Ao contrário dos outros dois amplificadores, este abrange todas as bandas de frequência geradas pelo sintetizador de frequência.

Do lado direito do diagrama é possível observar a existência de dois *buffers* de dados independentes, um para a modulação LoRa e outro para FSK/OOK, responsáveis pelo armazenamento da informação a transmitir ou recebida. Estes dois *buffers* de dados podem ser configurados e controlados través dos respetivos registos de configuração e de controlo. Por sua vez estes podem ser acedidos através de um interface SPI.

O módulo SX1276 possui ainda duas referência de tempo, um oscilador RC (Resistance-Capacitance) e um oscilador de cristal de 32 MHz. Ainda a partir da Figura 3.17 é possível observar a existência de 26 pinos. A Tabela 3.7 resume os pinos I/O mais relevantes ao nível do utilizador.

Tabela 3.7: Pinos I/O do módulo SX1276 [56].

Nome	Direção	Descrição
NReset	In	Pino de reset
NSS	In	Seletor de dispositivo SPI
MOSI	In	Entrada de dados SPI
MISO	Out	Saída de dados SPI
SCK	In	SPI clock
DIO0...5	In/Out	I/O Digital. Configurado por software
Vcc	In	Alimentação
GND	-	Terra

É importante referir que na Tabela 3.7, o Vcc engloba todos os pinos iniciados por VBAT ou VR representados na Figura 3.17.

3.5.2 Modem LoRa

Como já foi referido anteriormente, o módulo de rádio SX1276 integra dois *modems*, um LoRa e um FSK porém, apenas o modem LoRa será estudado uma vez que este é o objetivo do trabalho. No diagrama de blocos representado na Figura 3.18 é possível observar os diferentes constituintes do *modem* LoRa e FSK representados, respetivamente, na figura a rosa e a verde. A partir deste também é visível a existência de componentes em comum e outros independentes. Ambos os *modems* apresentam um *buffer* de dados FIFO (First In First Out) independente que pode ser acedido através dos registos de configuração, porém estes são partilhados por ambos os *modems*. Apesar de estes serem partilhados, o mapeamento da memória varia consoante o *modem* utilizado. Além disso, o

acesso aos registos de configuração é efetuado através de um interface SPI, também partilhado por ambos.

No capítulo seguinte, Capítulo 3.5.3, serão abordados com maior detalhe os diferentes constituintes do *modem* LoRa.

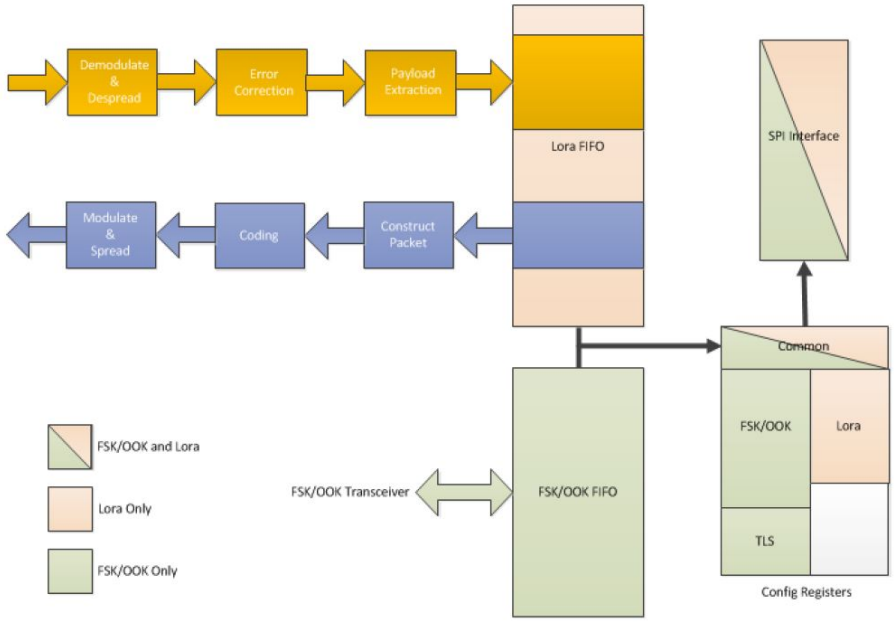


Figura 3.18: Diagrama de blocos do modem LoRa [56].

3.5.3 Interface Digital LoRa

O modem LoRa é constituído por três tipos de interfaces digitais, os registos estáticos de configuração, os registos de estado e um *buffer* de dados FIFO. Todos estes são acedidos através do interface SPI presente no módulo de rádio SX1276.

Os registos estáticos de configuração, como o nome indica, têm como objetivo configurar os diferentes parâmetros, tais como, os parâmetros referentes à modulação LoRa, da contituição do pacote LoRa, da amplificação do módulo de rádio, entre outros. Os registos de estados fornecem informações acerca do estado de funcionamento do dispositivo, como por exemplo no processo de transmissão ou receção de dados. O *buffer* de dados FIFO, de uma forma simples, é uma unidade de memória RAM que permite ao utilizador aceder aos dados recebidos ou colocar nesta os dados que pretende enviar. Os registos estáticos de configuração mais importantes serão abordados no secção 3.5.3.1 e o buffer de dados FIFO será mencionado com maior detalhe na secção 3.5.3.2. No que diz respeito aos registos de estado, alguns destes serão abordados ao longo destas secções.

Todos os registos abordados ao longo das próximas secções assim como outros registos relevantes encontram-se resumidos numa tabela no Anexo A.

Outro aspecto que é importante referir é que determinados registos apenas podem ser acedidos em determinados modos de operação. O *modem* LoRa permite funcionar em 8 modos diferentes no entanto apenas serão referidos os 5 modos mais importantes e encontram-se apresentados na Tabela 3.8. A escolha do modo de funcionamento, assim como a ativação do modem LoRa é realizado através do registo *RegOpMode*.

Tabela 3.8: Modos de operação do modem LoRa [56].

Modo de Operação	Descrição
Sleep	Modo de poupança de energia. Neste modo apenas registos de configuração pode ser acedidos. LoRa FIFO não pode ser acedida.
Standby	Os osciladores presentes no módulo encontram-se ligados. Componentes ligados à comunicação RF encontram-se desligados.
TX	Quando ativado todos os blocos necessários para a transmissão são ativados. Após a transmissão retorna ao modo Standby.
RXCONTINUOUS	Quando ativado todos os blocos necessários à receção são ativados. O modo apenas é alterado quando o utilizador pretender.
RXSINGLE	Quando ativado todos os blocos necessários à receção são ativados. Após a receção de um pacote válido, este retorna ao modo Standby.

3.5.3.1 Registos Estáticos de Configuração LoRa

Os registos estáticos de configuração permitem alterar diversos parâmetros de funcionamento do módulo SX1276, no entanto neste trabalho será dado maior ênfase aos parâmetros referentes à modulação e aos pacotes LoRa. Estes registos podem ser acedidos para leitura em qualquer modo de funcinamento, no entanto uma operação de escrita apenas pode ser realizada se o *modem* LoRa se encontrar nos modos de funcionamento *Sleep* ou *Standby*.

De forma a otimizar a modulação LoRa para uma aplicação específica, esta permite a alteração de três parâmetros críticos, nomeadamente o *spreading factor*, a *bandwidth* e o *coding rate*. Na modulação Lora é possível escolher de entre 7 valores de SF, de 6 a 12, no entanto o valor 6 apenas é utilizado em casos especiais quando se necessita da velocidade de transmissão mais rápida e que por vezes

este valor é descartado por vários autores. O SF pode ser alterado através do registo de configuração *RegModemConfig2*. Em relação ao CR este pode assumir quatros valores distintos, de 1 a 4, porém quanto maior for este valor maior a robustez na comunicação, mas também maior será o tempo de transmissão. A Tabela 3.9 demonstra a relação entre o CR, o CCR e o Overhead Ratio. O valor do CR pode ser alterado através do registo de configuração *RegModemConfig1*.

Tabela 3.9: Diferentes valores de CR disponíveis no modem LoRa [56].

Coding Rate	Cyclic Coding Rate	Overhead Ratio
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

Por último, é possível definir 10 valores diferentes para a *bandwidth*. Estes estão apresentados na Tabela 3.10 e são configurados através do registo *RegModemConfig1*. A partir da tabela é também possível observar a relação entre a BW e a taxa nominal de transmissão (R_b), em bps. Há medida que a BW aumenta, o (R_b) também aumenta.

Tabela 3.10: Diferentes valores para a BW disponíveis e a relação com a taxa de transmissão [56].

Bandwidth (kHz)	Spreading Factor	Coding Rate	Nominal Rb (bps)
7.8	12	4/5	18
10.4	12	4/5	24
15.6	12	4/5	37
20.8	12	4/5	49
31.2	12	4/5	73
47.1	12	4/5	98
62.5	12	4/5	146
125	12	4/5	293
250	12	4/5	586
500	12	4/5	1172

Em relação à estrutura do pacote LoRa, diversos parâmetros podem ser alterados a partir dos registos de configuração. A indicação da presença do *header* no pacote LoRa é um dos aspetos mais importantes e este pode ser efetuado através do *bit ImplicitHeaderModeOn* do registo *RegModemConfig1*. Além disto, outros parâmetros podem ser configurados como é o caso da existência do CRC no *payload* através do *bit RxPayloadCrcOn* no registo *RegModemConfig2*. Adicional-

mente, o comprimento do *preamble* e do *payload* podem ser configurados através dos respetivos registos, *RegPreambleMsb* / *RegPreambleLsb* e *RegPayloadLength*.

3.5.3.2 *Buffer* de Dados FIFO

O módulo de rádio SX1276 encontra-se equipado com uma memória RAM de 256 *bytes* reservada e unicamente acessível através da utilização do *modem* LoRa. Esta memória RAM, por vezes também referida por *buffer* de dados FIFO, é completamente personalizável e permite ao utilizador aceder ao dados recebidos, bem como os dados que serão tansmitidos. O acesso a esta memória é realizado através do interface SPI. A Figura 3.19 representa um diagrama de blocos do *buffer* de dados FIFO que demonstra o princípio de funcionamento da mesma. Antes de avançar é importante referir que a leitura dos dados nesta memória pode ser efetuada em qualquer modo de operação, excepto no modo *sleep*. Caso se pretenda escrever neste é recomendada a utilização o modo *standby*.

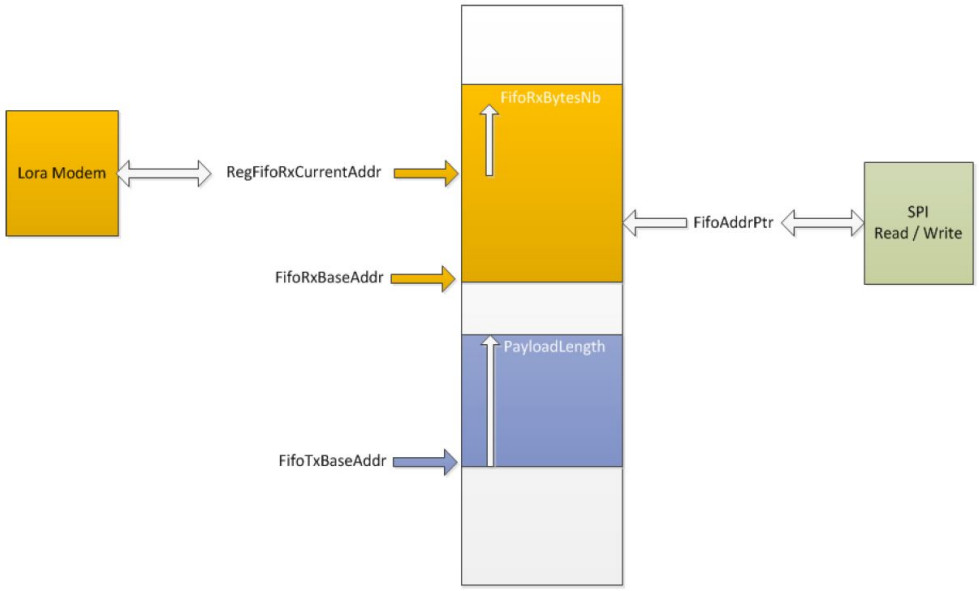


Figura 3.19: Princípio de funcionamento do *buffer* de dados FIFO [56].

Devido à configuração de duas portas do *buffer* de dados FIFO, é possível armazenar simultaneamente os dados recebidos e os que se pretende transmitir. O registo *RegFifoTxBaseAddr* específica a local da memória onde a informação transmitida é armazenada. Similarmente, para o processo de receção, o registo *RegFifoRxBaseAddr* indica o local onde a informação será armazenada caso se dê uma operação de receção de dados. Ao iniciar o dispositivo, por predefinição, metade da memória está alocada para operações de recepção, enquanto que a outra metade é dedicada para processos de transmissão. Desta forma o registo

RegFifoRxBaseAddr é inicializado com o valor 0x00 e o registo *RegFifoRxBaseAddr* é inicializado com o valor 0x80. No entanto, o valor destes endereços podem ser configurados de diversas formas ao longo dos 256 *bytes* de memória consoante o utilizador pretender.

O *buffer* de dados FIFO não pode ser acedido no modo de funcionamento *sleep* e é limpo sempre que o dispositivo é colocado neste modo de operação. Porém, toda a informação contida na memória é mantida caso o modem LoRa opere nos restantes modos de operação, a não ser que novos dados sejam sobrepostos a dados já existentes na memória.

A leitura ou escrita no *buffer* de dados FIFO é realizada via SPI através do registo *RegFifoAddrPtr*. Este registo serve como apontador para o endereço de memória onde se pretende ler ou escrever. Antes de qualquer leitura ou escrita, este apontador deve ser inicializado e, após a leitura ou escrita, este irá incrementar automaticamente.

Além dos registos já abordados, ainda existem outros registos que são importantes referir. O registo *RegRxNbBytes* indica o tamanho em *bytes* do último pacote de dados recebido. Por outro lado, o registo RegPayloadLength indica o tamanho de memória alocado pelos dados que serão transmitidos. Por último o registo *RegFifoRxCurrentAddr* indica a localização do último pacote recebido.

É importante referir que todos os dados recebidos são armazenado no *buffer* de dados FIFO mesmo que estes apresentem um CRC inválido, permitindo ao utilizador realizar um pós processamento dos dados corrompidos. Além disto, caso seja recebido um pacote com dimensão superir à memória alocada para este efeito, os dados recebidos irão sobrepor os dados armazenados para a operação de transmissão.

3.5.3.3 Interrupções e Pinos I/O

No modo de funcionamento LoRa são disponibilizados dois registos para o controlo das interrupções, o registo *RegIrqFlagsMask* que é utilizado para ativar ou desativar interrupções e o registo *RegIrqFlags* que indica quais as interrupções que foram geradas. No registo *RegIrqFlagsMask*, colocando o *bit* a 1 desativa a interrupção. No registo *RegIrqFlags* um *bit* a 1 indica que a interrupção foi ativada. Para limpar a flag é necessário escrever um 1 na respetiva flag. São disponibilizadas 8 interrupções, no entanto as de maior interesse encontram-se apresentadas na Tabela 3.11.

Tabela 3.11: Interrupções disponíveis no modo LoRa [56].

Interrupção	Descrição
TxDone	Indica que a transmissão de um pacote foi finalizada.
ValidHeader	Indica que foi recebido um pacote com um *header* válido.
PayloadCrcError	Indica um erro CRC na *payload*.
RxDone	Indica que a receção de um pacote foi finalizada.
RxTimeout	Indica que ocorreu um *timeout* para a receção de dados.

O módulo de rádio SX1276 disponibiliza 6 pinos I/O de uso geral. Estes possuem o mesmo objetivo das interrupções. Um pino com nível lógico alto indica que uma determinada interrupção foi ativada. A configuração dos pinos I/O pode ser realizado através dos registos *RegDioMapping1* e *RegDioMapping2* e consoante a sua configuração estes pinos podem assumir o mapeamento da Tabela 3.12.

Tabela 3.12: Mapeamento dos pinos I/O e as respetivas interrupções [56].

DIOx Mapping				
	00	**01**	**10**	**11**
DIO0	RxDone	TxDone	CadDone	-
DIO1	RxTimeout	FssChangeChannel	CadDetected	-
DIO2	FssChangeChannel	FssChangeChannel	FssChangeChannel	-
DIO3	CadDone	ValidHeader	PayloadCrcError	-
DIO4	CadDetected	PIILock	PIILock	-
DIO5	ModeReady	ClkOut	ClkOut	-

3.5.3.4 Processo de Transmissão de Dados

O fluxograma apresentado na Figura 3.20 demonstra a sequência de passos necessários para efetuar uma transmissão. A transmissão inicia sempre com a configuração dos registos estáticos necessários, bem como o preenchimento do *buffer* de dados FIFO com os dados que se pretende enviar. Estes dois passos são realizados através dos respetivos blocos *Tx_Init* e *Write_Data_FIFO* do fluxograma. Porém, estes apenas podem ser acedidos no modo de operação *standby*, daí ser necessário primeiramente colocar o dispositivo neste modo de operação. A transmissão é iniciada após a alteração do modo de operação para Tx, executada no bloco *Mode_Request_Tx*. Quando a transmissão terminar será executada a interrupção TxDone e o *modem* LoRa entra automaticamente em modo *standby*. Após isto, o utilizador pode optar por escolher um novo modo de operação ou, caso possua mais dados para enviar, preencher novamente a memória de dados FIFO e repetir todo o processo.

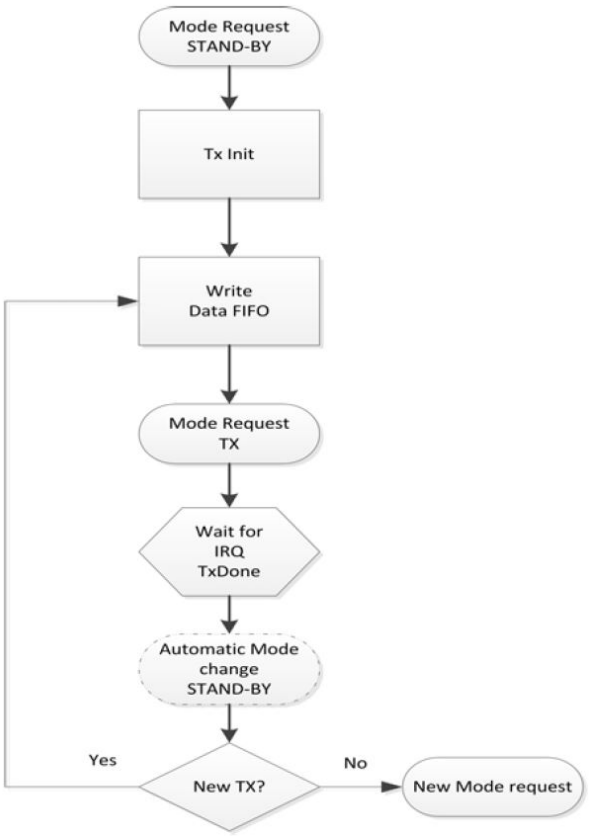


Figura 3.20: Fluxograma do processo de transmissão de dados [56].

De forma a colocar o pacote de dados a transmitir na memória de dados FIFO, representado pelo bloco *Write_Data_FIFO* no fluxograma, é necessário colocar o valor que se encontra no registo *RegFifoTxBaseAddr* no registo *RegFifoAddrPtr* e de seguida colocar os dados que se pretende enviar no registo *RegFifo*.

3.5.3.5 Processo de Receção de Dados

O fluxograma apresentado na Figura 3.21 explica todo o processo que envolve a receção de dados. Através da observação deste fluxograma é possível observar que existem dois modos para a receção de dados, o modo *Rx_Single* e o *Rx_Continuous*.

Similarmente ao processo de transmissão, a receção inicia com a configuração dos registos necessários e com a preparação do *buffer* de dados FIFO para a receção de dados e estas etapas, assim como na transmissão, apenas podem ser executadas se o *modem* LoRa estiver no modo de operação *standby*. De seguida é escolhido o modo de funcionamento e, caso seja utilizado o modo de receção contínuo, o dispositivo é colocado no modo *Rx_Continuous*. A receção de um

pacote com um CRC no *header* válido gera a interrupção *RxDone*, que indica a receção de um pacote. Caso a *payload* possua CRC, a verificação desta deve ser avaliada. Caso a integridade do pacote não tenha sido comprometida este pode ser lido da memória de dados FIFO. Enquanto que o utilizador não alterar o modo de operação do *modem* este repetirá continuamente os passos anteriores.

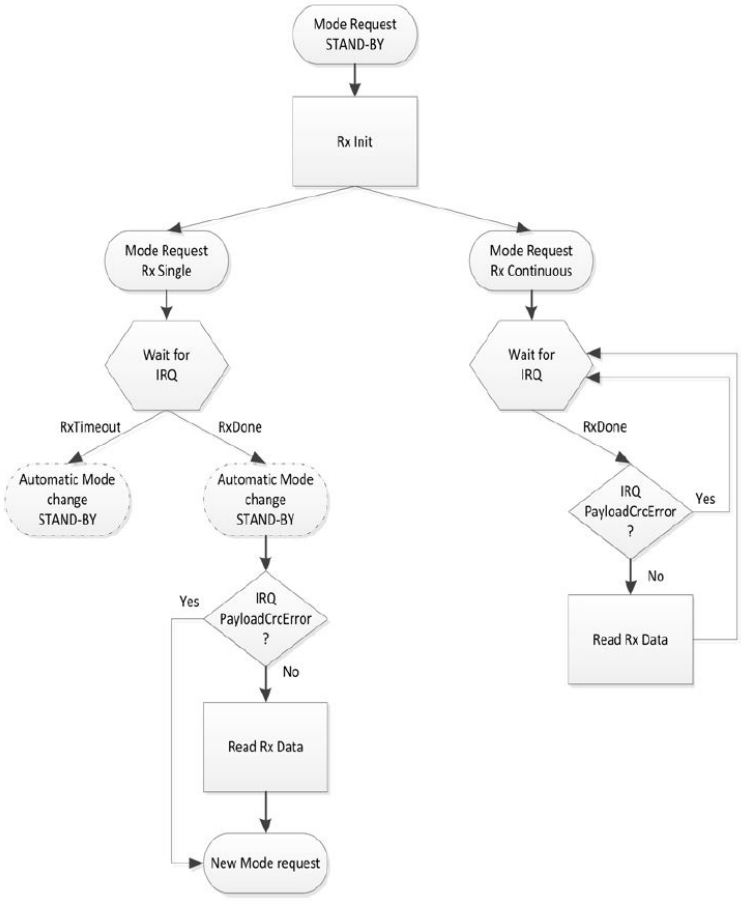


Figura 3.21: Fluxograma dos processos de receção de dados [56].

Caso se pretenda receber uma única mensagem é utilizado o modo *Rx_Single*. Assim como no modo *Rx_Continuous*, a receção de um pacote com um CRC no *header* válido gera a interrupção *RxDone*, que indica a receção de um pacote. Porém, após a receção, o dispositivo é colocado automaticamente em modo *standby*. De seguida é validado o CRC da *payload*, caso exista, e é extraída a mensagem da FIFO. Posteriormente é ainda especificado o novo modo de funcionamento. Porém, neste modo existe um período de tempo no qual é possível receber uma mensagem. Este período é designado de *timeout* e é definido através do registo *RegSymbTimeoutLsb* e pelos dois *bits* menos significativos do registo

RegModemConfig2. Isto indica que este valor, designado por *SymbTimeout*, pode variar entre 0 e 1023. Para o cálculo do *timeout* é utilizada a expressão 3.3.

$$Timeout = SymbTimeout \cdot T_{Symbol} = SymbTimeout \cdot \frac{2^{SF}}{BW} \tag{3.3}$$

Caso não seja recebida nenhuma mensagem neste período de tempo, é gerada a interrupção *RxTimeout* e o dispositivo é colocado automaticamente em modo *standby*.

A extracção de dados da memória é semelhante ao da transmissão. O registo *RegFifoRxCurrentAddr* indica o endereço onde foi recebido a última informação. Colocando este endereço no registo *RegFifoAddrPtr*, pode se iniciar a leitura do registo *RegFifo*, *RegRxNbBytes* vezes, uma vez que este indica tamanho em *bytes* do pacote recebido.

Capítulo 4

Arquitetura do Sistema

O objetivo deste capítulo passa por apresentar a arquitetura da solução implementada, bem como descrever e analisar os diferentes constituintes.

4.1 Diagrama Geral da Solução

Na Figura 4 encontra-se representado um diagrama de blocos generalizado do sistema implementado. Este sistema tem como objetivo recriar um sistema *healthcare* simples baseado na filosofia da IoT e teve por base o esquema apresentado na Figura 2.14 do capítulo 2.5.1.

Este sistema é composto por dois nós. O primeiro encontra-se equipado com um atuador e com um sensor e é responsável pela captação da informação. O segundo serve como intermediário entre o nó 1 e o servidor (representado na Figura 4 como Internet), tendo como objetivo não só de enviar a informação recebida do nó 1 para o servidor, mas também as mensagens de controlo do atuador do servidor para o nó 1. Além dos dois nós, o sistema possui um servidor simples capaz de gerir a página *web*, a bases de dados e a troca de mensagens com o nó 2.

De modo a facilitar a análise da solução implementada esta será dividida em três etapas distintas. A primeira diz respeito ao sistema de comunicação LoRa e tem como objetivo o estabelecimento da comunicação LoRa entre os dois nós. A segunda etapa refere-se à comunicação entre o nó 2 e o servidor através de uma ligação Wi-Fi. Por último, numa terceira fase, pretende-se abordar a ligação existente entre o servidor, a base de dados e a página *web*. Em cada umas destas fases serão referidas as escolhas de *hardware*, assim como as ferramentas utilizadas para o desenvolvimento do *firmware* e *software*.

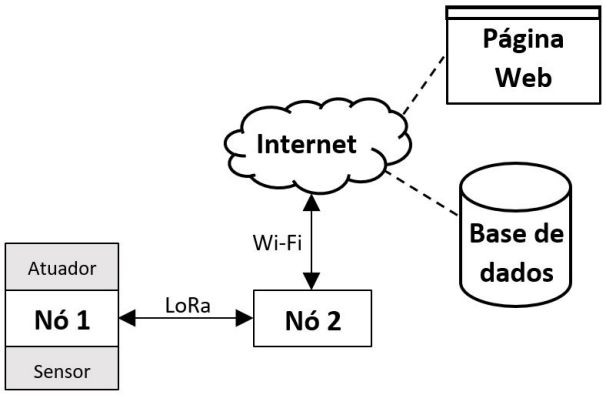


Figura 4.1: Diagrama de blocos da arquitetura do sistema.

4.2 Sistema de Comunicação LoRa

O diagrama de blocos da Figura 4.2 representa o sistema de comunicação LoRa de uma forma mais detalhada. Este sistema de comunicação será utilizado entre os nós 1 e 2. Além disso, é importante referir que este se trata de sistema de comunicação bidirecional, logo, cada um dos nós opera como transceptor. De forma a utilizar uma terminologia mais adequada o nó 1 e nó 2 passarão a ser referidos como *end device* e *gateway*, respetivamente. Ambos os transceptores serão analisados individualmente devido ao facto de possuírem diferentes objetivos e consequentemente componentes particulares, apesar de estes apresentarem alguns aspetos em comum.

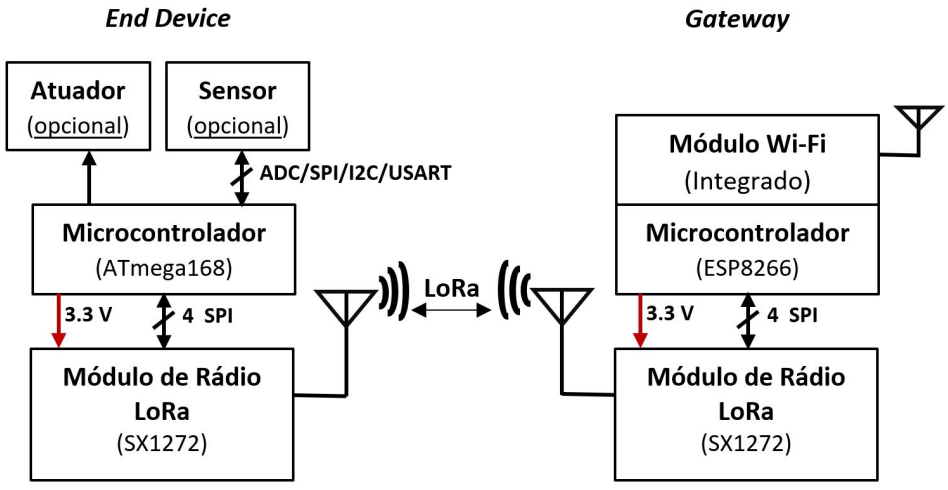


Figura 4.2: Diagrama de blocos do sistema de comunicação LoRa.

4.2.1 *End Device*

O *end device* apresenta a composição típica de um nó LoRa. Este encontra-se equipado com módulo de rádio LoRa e por um microcontrolador. Associado a estes, o transceptor integra ainda um sensor e um atuador. Além disso, este transceptor utiliza uma alimentação externa, o que condicionará a escolha do microcontrolador uma vez que se pretende prolongar a duração da bateria. Deste modo, optou-se por utilizar um ATmega168 da gama de 8 bits da Atmel. Este é um microcontrolador de baixo consumo no entanto com capacidade de processamento suficiente para a aplicação em questão. Este é responsável pelo controlo do sensor, do atuador e pelo controlo do módulo de rádio LoRa. A Tabela 4.1 resume as caraterísticas principais do microcontrolador ATmega168. Na Figura 4.3 é apresentado o *pin mapping* do mesmo.

Tabela 4.1: Características do microcontrolador ATmega168 [86].

Características do ATmega168	
Arquitetura RISC	3 timer/counters
Memória flash de 16 kB	Interrupções externas e internas
512 bytes de EEPROM	USART/SPI/I2C
1 kB de SRAM	Conversor A/D de 10 bits
23 linhas I/O programáveis	Frequência máxima de 20 MHz

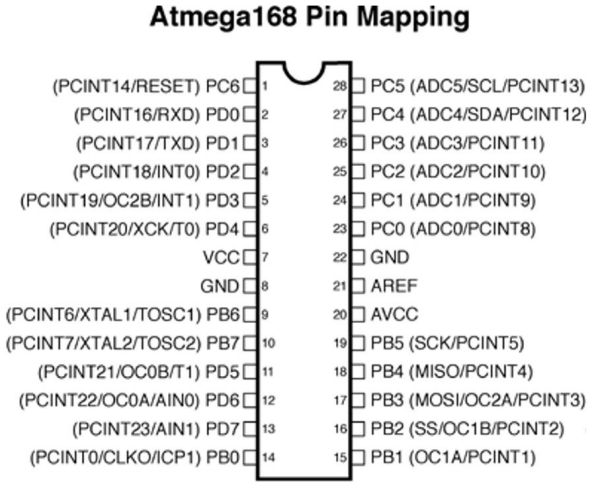


Figura 4.3: *Pin mapping* do ATmega168. Adaptado de [87].

Para a comunicação com o módulo de rádio LoRa é utilizado o protocolo SPI. Relativamente ao módulo de rádio LoRa, este será abordado mais à frente na secção 4.2.3 uma vez que é utilizado em ambos os transceptores. No entanto é importante referir nesta secção que este módulo possui diferentes modos de

operação. Dentro destes, destaca-se o modo *sleep* pela sua importância ao nível da gestão energética do próprio transceptor, devido ao tipo de alimentação que este apresenta. Outro aspeto importante é a alimentação do ATmega168 e consequentemente de todo o transceptor. O facto de o módulo rádio não suportar tensões de alimentação, nos pinos digitais e analógicos superiores a 3.9 V, implica que a alimentação do microcontrolador seja de 3.3 V. De forma a baixar a tensão para os níveis desejados é utilizado um circuito regulador de tensão que permite baixar a tensão de alimentação externa para os valores pretendidos.

Para a programação do ATmega168 é utilizada a linguagem de programação C, com recurso ao *software* Atmel Studio 7 [88].

4.2.2 *Gateway*

Assim como o *end device*, a *gateway* possui um microcontrolador e um módulo de rádio LoRa. Porém, como o objetivo deste é servir como intermediário entre o *end device* e o servidor, este necessita de possuir outra tecnologia de comunicação sem fios, o Wi-Fi. Além desta imposição, a unidade de controlo terá de respeitar os requisitos impostos pelo módulo de rádio LoRa, nomeadamente as tensões de alimentação e o protocolo de comunicação SPI.

Ponderados estes requisitos, foi selecionado para unidade de controlo deste transceptor o NodeMCU ESP8266. Esta é uma placa de desenvolvimento constituída pelo SoC ESP8266EX que integra o sistema de comunicação Wi-Fi e o processador Tensilica L106 de 32 bit. Outro aspeto importante desta placa de desenvolvimento é o facto de possuir um ciruito regulador de tensão para 3.3 V integrado, assim como uma antena Wi-Fi embutida e um interface USB. Esta é uma caraterística importante uma vez que este transceptor terá uma alimentação continua e será efetuada através da entrada USB [89]. A Tabela 4.2 resume as principais caraterísticas do ESP8266.

Tabela 4.2: Características do SoC ESP8266EX [90].

Características do SoC ESP8266EX	
CPU: Tensilica L106 32 bit	802.11 b/g/n Wi-Fi: HT20
Frequência máxima de 80 MHz	IPv4, TCP/UDP/HTTP
16 Mbytes de memória flash	USAT/SPI/SPIO/I2C/I2S
160 kB de SRAM	Conversor A/D de 10 bits
17 GPIO	PWM: 8 canais

Na Figura 4.4 é possível observar o *pin mapping* da placa de desenvolvimento. Assim como para o *end device*, neste também será utilizado o protocolo de comunicação SPI para comunicar com o módulo de rádio LoRa que será abor-

dado na secção 4.2.3. Para a programação do ESP8266 é utilizada a linguagem de programação C++, através do Arduino IDE.

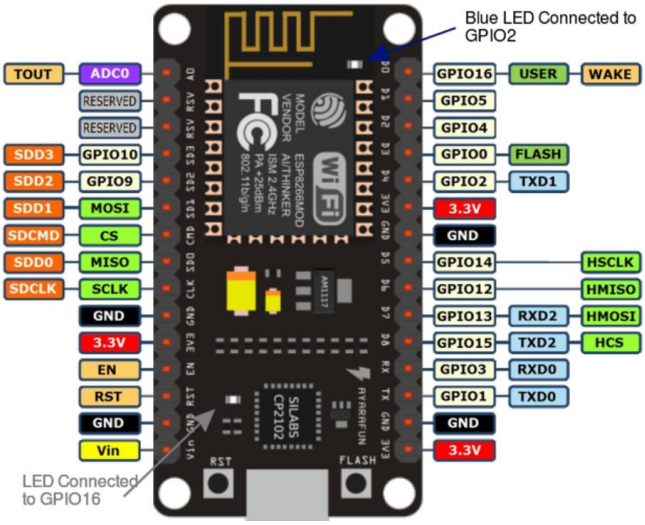


Figura 4.4: *Pin mapping* do NodeMCU ESP8266 [89].

4.2.3 Módulo de Rádio LoRa

O módulo de rádio escolhido para a transferência de mensagens entre os dois transceptores foi o RF-LoRa da RF Solutions [91]. Este é um módulo que se encontra equipado com o *chip* SX1272 da Semtech que opera na banda de frequências ISM europeia, 868 MHz, e permite comunicações até 16 km [92].

O *pin mapping* do módulo de rádio RF-LoRa encontra-se apresentado na Figura 4.5. O princípio de funcionamento é idêntico ao do SX1276 abordado na secção 3.5, apresentando apenas algumas diferenças no mapa de memória.

Como já foi referido anteriormente, a comunicação com o módulo de rádio LoRa é efetuada através do protocolo de comunicação SPI. Para esse efeito são utilizados os pinos nSEl, SDI, SDO e SCLK. Os pinos designados de DIOx têm como objetivo indicar a ocorrência de alguma interrupção que tenha ocorrido durante o processo de transmissão ou receção, como já foi referido na secção 3.5.3.3. É importante salientar que a sua utilização não é obrigatória para o correto funcionamento do módulo.

Por outro lado, os pinos RX_SWITCH e TX_SWITCH são essenciais para o funcionamento do módulo. É a partir destes que é selecionado o modo de funcionamento do módulo de rádio LoRa, isto é, se ele irá operar como um transmissor ou como um recetor. Caso se pretenda colocar no modo de transmissão, o pino TX_SWITCH deverá ser colocado a 1 e o RX_SWITCH a 0, e vice-versa

caso se pretenda utilizar o modo de receção. No pino ANT deve ser conectada a antena do módulo. É importante referir que a alimentação do módulo sem nenhuma antena conectada poderá danificá-lo.

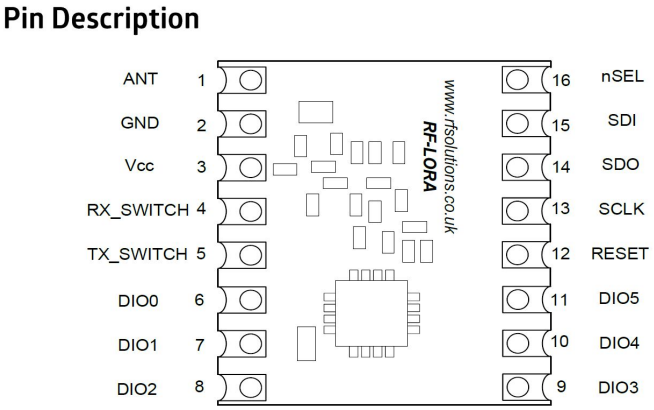


Figura 4.5: *Pin mapping* do módulo de rádio RF-LoRa [91].

Por último, e como já foi referido anteriormente, o módulo não suporta em nenhum pino tensões superiores a 3.9 V sendo recomendado a utilização de uma tensão de 3.3 V.

4.3 Sistema de Comunicação com o *Back-End*

Na Figura 4.6 é apresentado o diagrama de blocos correspondente ao sistema de comunicação com o *back-end*, mais propriamente a comunicação entre a *gateway* e o servidor através do Wi-Fi. Para o envio de mensagens entre estes dois componentes é utilizado o protocolo de comunicação MQTT.

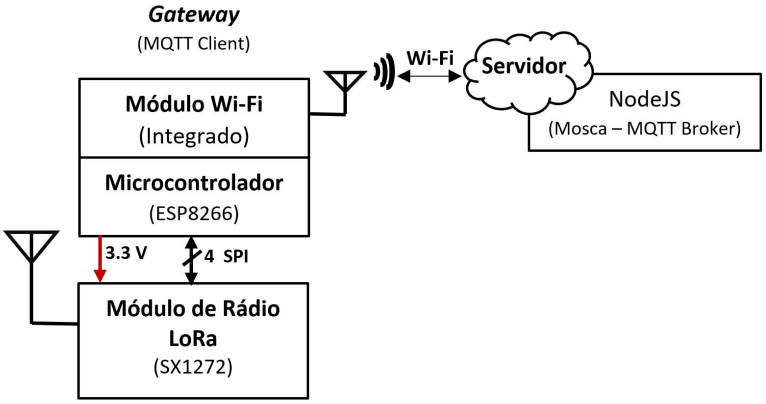


Figura 4.6: Diagrama de blocos do sistema de comunicação com *back-end*.

O MQTT é um protocolo leve desenvolvido no contexto da IoT. Além disso é um protocolo assíncrono, com uma arquitetura *publish/subscribe* e que tem por base o protocolo de transporte TCP.

A arquitetura *publish/subscribe* é uma das princípais carateristicas que define o protocolo MQTT e que assenta em dois conceitos fundamentais: o conceito *publish/subscribe* e o conceito de tópicos e subscrições. No que diz respeito ao conceito *publish/subscribe*, no protocolo MQTT uma entidade pode assumir o papel de *publisher* ou *subscriber*, isto é, caso atue como um publisher, este será responsável por enviar mensagens e, caso assuma o papel de *subscriber*, este será responsável por receber mensagens. Relativamente ao conceito de tópicos e subscrições, um tópico pode ser visto como o assunto de uma mensagem. Isto possibilita a existência de mensagens específicas às quais uma entidade pode subscrever e deste modo receber apenas as mensagens de determinados tópicos. Existe uma forte interligação entre estes dois conceitos sendo por vezes difícil a compreensão quando referidos de forma independente. Combinando estes, e em modo de resumo, um *publisher* tem como função publicar mensagens em determinados tópicos, enquanto que, um *subscriber* subscreve a determinados tópicos, recebendo assim apenas as mensagens referentes a estes.

Além destes dois conceitos fundamentais, a arquitetura do protocolo MQTT pode ser dividida em dois componentes principais: clientes e o *broker*. Um cliente pode assumir o papel de *publisher* ou *subscriber* e estabelece sempre uma ligação com o servidor (*broker*). Relativamente ao *broker*, este tem como função gerir toda a informação tendo como principal responsabilidade filtrar todas as mensagens enviadas pelos *publishers* e reencaminhá-las para os clientes subscritos nos respetivos tópicos [93].

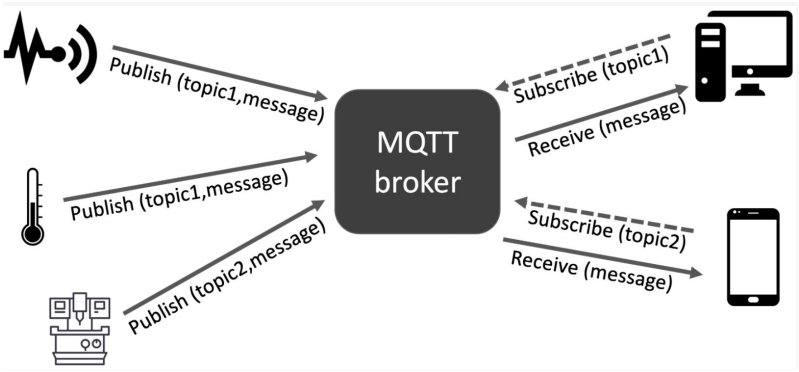


Figura 4.7: Princípio de funcionamento do protocolo MQTT [94].

Como já foi referido, o *broker* desempenha um papel vital no protocolo MQTT uma vez que serve de elo de ligação entre a aplicação e o ambiente físico, e está diretamente relacionado com o seu desempenho. Actualmente existem diversos

brokers disponíveis no mercado, cada um com determinados aspetos positivos e negativos. Neste trabalho optou-se pela utilização do *broker* Mosca. O aspeto fundamental para a escolha do Mosca é o facto de este poder ser utilizado juntamente com outras aplicações desenvolvidas através do NodeJS. Este foi o aspeto fundamental uma vez que o *front-end* também será desenvolvido com recurso ao NodeJS como será referido de seguida na secção 4.4.

4.4 Ligação entre *Back-End* e *Front-End*

O diagrama de blocos da Figura 4.8 pretende representar a constituição e as conexões existentes entre o *front-end* e o *back-end*.

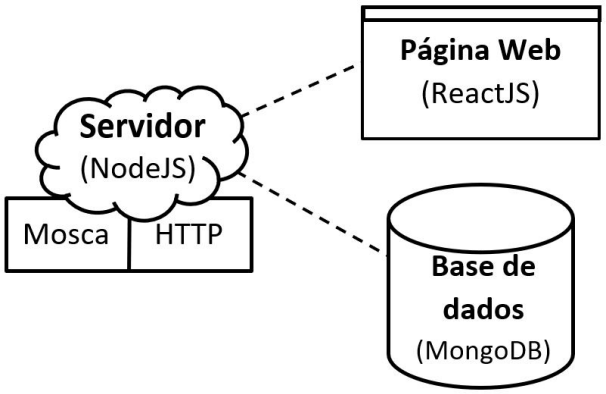


Figura 4.8: Diagrama de blocos referente à constituição do *back-end* e *front-end* e respetiva ligação.

4.4.1 *Back-End*

O *back-end* é constituido pelo servidor e pela base de dados e tem como objetivo gerir a informação proveniente do ambiente físico, armazená-lo numa base de dados e permitir a demonstração da mesma numa página web (designada de *front-end*).

De uma forma simples o servidor pode ser dividido em duas partes, uma parte responsável pela comunicação com a *gateway*, já referida na secção 4.3 e outra responsável pela comunicação com a base de dados e com a *front-end*. O *back-end* será desenvolvido com recurso ao NodeJS [95]. Para efetuar a ligação entre o *back-end*, a base de dados e o *front-end* é utilizado o protocolo de comunicação HTTP, implícito numa *framework* do NodeJS denominada de Express [96]. Para a armazenar toda a informação será utilizada a base da dados MongoDB [97].

A lista seguinte pretende abordar de forma sucinta as tecnologias referidas, nomeadamente o NodeJS, Express e MongoDB, de modo a entender o que são e a sua importância no *back-end*.

- **NodeJS** é um interpretador de JavaScript assíncrono *open source* com código orientado a eventos. Este pode ser utilizando a nível do *front-end* e *back-end* consoante as bibliotecas e *frameworks* utilizadas.
- **Express** é uma *framework* minimalista e flexível do NodeJS que fornece um conjunto robusto de recursos para desenvolver aplicações web e móveis. Este permite configurar *middlewares* para responder a pedidos HTTP. Além disto permite definir uma tabela de roteamento que é usada para realizar diferentes ações com base no método HTTP e URL. Estas são duas das caraterísticas essenciais da *framework* Express.
- **MongoDB** é uma base de dados orientada a documentos que fornece alto desempenho, alta disponibilidade e fácil escalabilidade. Esta trabalha sobre o conceito de coleções e documentos. Um coleção é a designação que se dá a um grupo de documentos. Por sua vez um documento é onde é colocada a informação que se pretende armazenar e apresenta uma estrutura dinâmica, isto é, cada documento de uma coleção pode possuir estruturas diferentes. No entanto, por uma questão de boa prática, isto não acontece sendo por norma armazenado numa coleção documentos com estruturas idênticas. Dentro de cada documento os dados são armazenados no formato *JavaScript Object Notation* (JSON).

Todo o código referente ao *back-end* e *front-end* será desenvolvido em JavaScript através do ambiente de desenvolvimento Visual Studio Code [98].

4.4.2 *Front-End*

A partir do *front-end* pretende-se observar os dados obtidos do sensor e controlar o atuador. O *front-end* do sistema irá ser desenvolvido em ReactJS [99]. Esta é uma biblioteca JavaScript *open source* para o desenvolvimento de interfaces. O seu principal objetivo é ser rápida, escalável e simples, podendo ainda ser usada em combinação com outras bibliotecas ou *frameworks* de JavaScript.

4.5 Arquitetura do Sistema

Em forma de resumo, a Figura 4.9 representa a arquitetura final do sistema. Além dos diferentes constituintes, este tenta englobar as diferentes tecnologias envolvidas na solução apresentada.

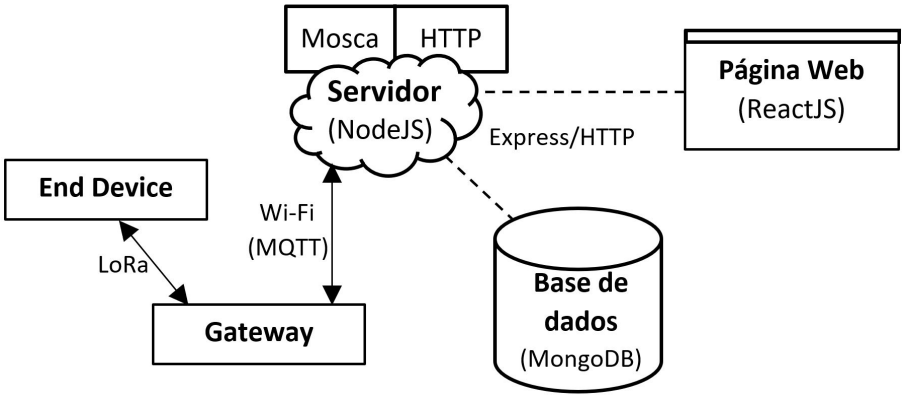


Figura 4.9: Arquitetura final do sistema.

Capítulo 5

Implementação do Sistema

Neste capítulo pretende-se abordar a implementação do sistema ao nível de *hardware*, *firmware* e *software*. Numa primeira fase é referida a parte de *hardware* onde são referidas as ligações entre os diferentes componentes. Numa segunda fase é exposto o *firmware* e *software* desenvolvidos nas diferente etapas do trabalho.

5.1 Sistema de Comunicação LoRa

Ao nível de *hardware* o sistema de comunicação LoRa é constituído por dois transceptores de rádio. Devido ao objectivo que cada um destes apresenta a sua constituição é diferente. Como já foi referido na secção 4.2.1, o *end device* é composto por um ATmega168 e pelo módulo de rádio RF-LoRa, juntamente com um sensor e um atuador. De modo a simulá-los será utilizado um potenciómetro e um LED, respetivamente. Adicionalmente, serão utilizados dois LED para indicar visualmente a transmissão e receção de uma mensagem através do módulo de rádio LoRa. Relativamente à *gateway*, esta apresenta uma constituição semelhante no entanto não possui nenhum atuador nem sensor e sua unidade de controlo é substituída por um NodeMCU ESP8266.

5.1.1 Esquema Elétrico e Configuração do Protocolo SPI

Na Figura 5.1 são apresentados os esquemas elétricos de ambos os transceptores. A Tabela 5.1 pretende complementar os esquemas elétricos. Nesta são referidas todas as ligações efetuadas entre os diferentes componentes presentes no *end device* e na *gateway*, bem como referir a sua configuração na respetiva unidade de controlo.

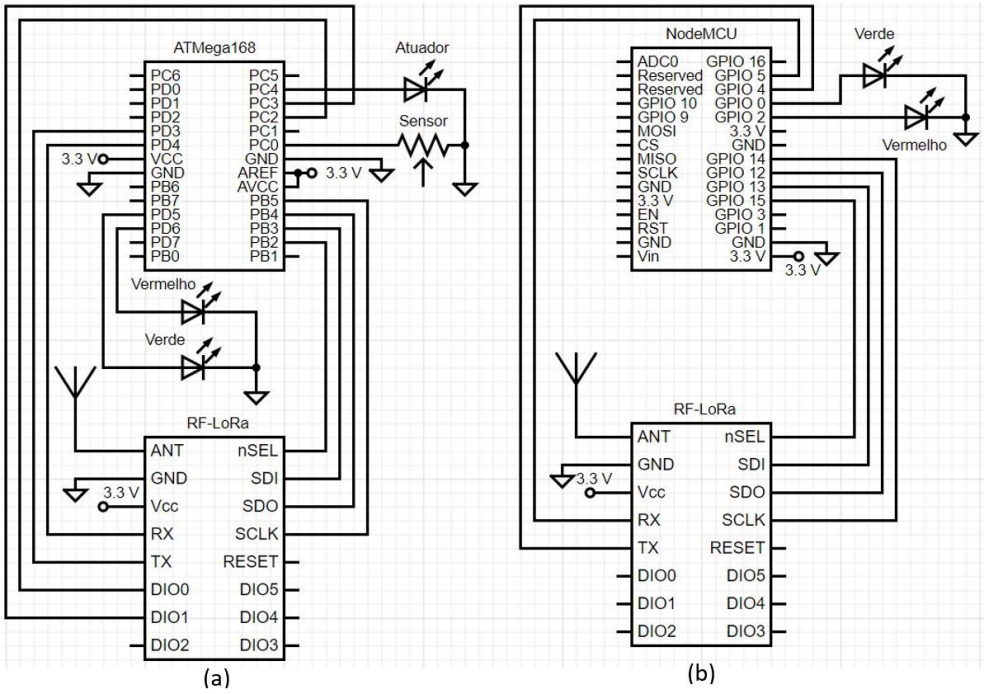


Figura 5.1: Circuitos implementados: (a) *End Device* e (b) *Gateway*.

Tabela 5.1: Tabela de ligações do *End Device* e da *Gateway*.

ATmega168	ESP8266	I/O	LoRa	LED/ Potenciómetro
PD4	GPIO5	Out	Rx_Switch	–
PD3	GPIO4	Out	Tx_Switch	–
PB5	GPIO14	Out	SCLK	–
PB4	GPIO12	In	SDO	–
PB3	GPIO13	Out	SDI	–
PB2	GPIO15	Out	nSEL	–
PC2	–	In	DIO0	–
PC3	–	In	DIO1	–
PD5	GPIO0	Out	–	Verde (Transmissão)
PD6	GPIO2	Out	–	Vermelho (Receção)
PC4	–	Out	–	Vermelho (Actuador)
PC0	–	ADC0	–	Potenciómetro

Uma vez que o *end device* apresenta uma alimentação externa foi ainda necessário utilizar um circuito regulador de tensão de modo a baixar a tensão da bateria para 5 V. Posteriormente foi utilizado um divisor de tensão para baixar esta tensão de 5 V para 3.3 V. Optou-se por reduzir a tensão para 5 V a partir do circuito regulador de tensão e só depois para 3.3 V através de um divisor

de tensão uma vez que, na eventualidade da utilização de um sensor ou atuador que necessite de uma alimentação de 5 V, esta já se encontrar disponível. Utilizaram-se valores baixos para as resistências do circuito regulador de tensão de modo a preservar a corrente proveniente da bateria. O esquema elétrico do circuito desenvolvido encontra-se apresentado na Figura 5.2.

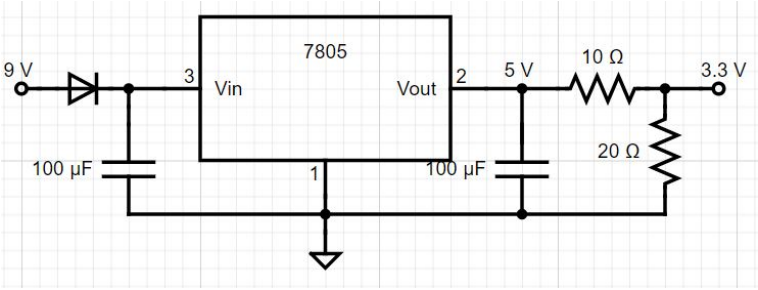


Figura 5.2: Circuito regulador de tensão.

No que diz respeito ao *firmware* desenvolvido para o sistema de comunicação LoRa este pode ser dividido em 3 componentes fundamentais: a configuração do módulo LoRa, as respetivas funções necessárias para transmissão e receção de mensagens e o princípio de funcionamento de ambos os transceptores. No entanto, é necessário primeiramente configurar o periférico SPI de modo a permitir a comunicação com o modúlo LoRa e efetuar a sua respetiva configuração.

Para efetuar a configuração do periférico SPI foi desenvolvida a função **SPI_Config()**, que encontra-se apresentada no extrato de código seguinte. Nesta função, e numa primeira fase, são definidos os pinos I/O **CS**, **MOSI**, **SCK** como saída e o pino **MISO** como entrada (linha 2). De seguida, é configurado o registo **SPCR** onde é ativado o periférico, é indicado que este operará como *master* e é definida a frequência do **SCK** (linha 3). Neste caso foi utilizado a frequência mais rápida disponível através da colocação dos *bits* **SPR1** e **SPR2** a zero. Para obter esta frequência é ainda necessário ativar o *bit* **SPI2X** no registo **SPSR** que permite duplicar a velocidade (linha 4). Deste modo, e uma vez que o microcontrolador opera a uma frequência de 8 MHz, a frequência do periférico SPI será de 4 MHz. Por último é chamada a função **UnselectReceiver()** (linha 5). Esta função coloca o pino **CS** a um nível lógico alto. Isto porque uma transição deste pino de um nível lógico alto para baixo indica o início de uma transmissão. Desta forma, ao colocar o pino **CS** num nível lógico alto, o periférico SPI permanece num estado de espera. Para a colocação do pino CS num nível lógico baixo foi desenvolvida a função **SelectReceiver()**.

```
1  void SPI_Config() {
     // set CS, MOSI and SCK to output
2    SPI_DDR |= (1 << CS) | (1 << MOSI) | (1 << SCK);
     // enable SPI, set as master, and clock to fosc/2
3    SPCR = (1 << SPE) | (1 << MSTR);
     // enable double SPI speed bit
4    SPSR = (1 << SPI2X);
     // set CS high, set low to transmit
5    UnselectReceiver();
   }
```

Posteriormente à configuração do periférico SPI foram desenvolvidas duas funções fundamentais responsáveis pela transferência de dados através deste periférico, nomeadamente **SPI_ReadRegister()** e **SPI_WriteRegister()**. A primeira foi desenvolvida para efetuar a leitura de um registo, enquanto que a segunda tem como obejtivo efetuar a escrita num registo. O excerto de código seguinte diz respeito à função **SPI_ReadRegister()**. Esta função recebe como parâmetro o endereço do registo que se pretende ler (linha 1). Este valor é colocado no registo de dados, **SPDR** (linha 2), e é esperado pelo seu envio (linha 3). Após o envio do endereço é enviado um *dummy byte*, neste caso 0xFF, que tem como objetivo "empurrar" a informação do contida no registo que se pretendia ler para o *master* (linha 4 e 5). Posteriormente o valor contigo no registo **SPDR** é retornado (linha 6).

```
1  uint8_t SPI_ReadRegister(uint8_t addr){
    // send address
2   SPDR = addr;
    // Wait for reception complete
3   while(!(SPSR & (1 << SPIF)));
    // transmit dummy byte
4   SPDR = 0xFF;
    // Wait for reception complete
5   while(!(SPSR & (1 << SPIF)));
    // return Data Register
6   return SPDR;
   }
```

A função **SPI_WriteRegister()** é semelhante a esta no entanto, em vez de ser enviado um *dummy byte* é enviado o valor que se pretende escrever nesse registo. Além disso, no final não existe a necessidade de retornar o valor contido no registo **SPDR**. As restantes funções abordadas, porém não expostas, relativas ao periférico SPI encontram-se no Anexo B.

5.1.2 Configuração do Módulo de Rádio LoRa

Como já foi referido, o módulo de rádio LoRa utiliza o *chip* SX1272 da Semtech. Este é na sua generalidade idêntico ao SX1276 já analisado na secção 3.5, apresentando apenas algumas diferenças no mapa de memória. Estes módulos permitem a configuração de diversos parâmetros, nomeadamente os parâmetros relativos à modulação LoRa e à amplificação do sinal.

Porém, de forma a ser possível configurar os diferentes parâmetros foi necessário desenvolver duas funções que permitiram aceder aos diferentes registos presentes no módulo de rádio LoRa, nomeadamente **LoRa_ReadRegister()** e **LoRa_WriteRegister()**. Estas funções têm por base as funções desenvolvidas para a transferência de dados através do interface SPI abordadas na secção 5.1.1. O extrato de código seguinte diz respeito à função **LoRa_ReadRegister()**. Esta recebe como parâmetro o endereço do registo que se pretende ler (linha 1). De seguida é chamada a função **SelectReceiver()** que permite iniciar uma transmissão através do interface SPI (linha 3). Posteriormente é chamada a função **SPI_ReadRegister()** que permite efetuar a leitura do registo pretendido (linha 4). É importante salientar que quando se pretende efetuar a leitura de um registo o *bit* mais significativo do endereço deve ser colocado a 0 e, caso se pretenda escrever, este deve ser colocado a 1. Terminada a leitura o valor é armazenado na variável **spi_buffer** (linha 4). Por último é chamada a função **UnselectReceiver()** que termina a transmissão através do periférico SPI, sendo retornado o valor da leitura (linha 5 e 6).

```
1  uint8_t LoRa_ReadRegister(uint8_t RegAddr){
2     uint8_t spi_buffer = 0;
3     SelectReceiver();
4     spi_buffer = SPI_ReadRegister(RegAddr & 0x7F);
5     UnselectReceiver();
6     return spi_buffer;
   }
```

A função desenvolvida para a escrita é semelhante à de leitura porém esta possui mais um parâmetro, mais concretamente o valor que se pretende colocar no registo. Além disso, utiliza a função **SPI_WriteRegister()** e não retorna nenhum valor. A função **LoRa_WriteRegister()** encontra-se apresentada no Anexo C.

Através destas funções foi possível configurar o módulo de rádio LoRa. Para esse efeito foi desenvolvida a função **LoRa_Config()**. Nesta secção apenas serão abordadas configurações mais importantes, no entanto no Anexo C é apresentada uma lista com todas a configurações efetuadas assim como o código completo desta função.

Ao nível da modelação LoRa ambos os transceptores foram configurados para operar a uma frequência de 869.500 MHz. Como se pretende que o alcance seja maximizado foi utilizado um *spreading factor* de 11 e uma *bandwith* de 125 kHz. Apesar de um SF de 12 permitir um maior alcance, optou-se por não utilizar uma vez que a combinação destes dois parâmetros torna a taxa de transmissão bastante baixa. Como se pretendia uma comunicação robusta optou-se por utilizar o *coding rate* de 4/8 porém o *overhead* na comunicação também é maior e consequentemente o período de transmissão. Em relação ao pacote Lora foi utilizado um *header* explícito e foi ativado o CRC para a *payload*. Foi ainda definido um tamanho para o *preamble* de 12.25 *symbols* e um tamanho máximo para a *payload* de 255 *bytes*. Ao nível da amplificação do sinal, e como se pretendia obter o maior alcance possível, foi utilizado como pino de saída o PA_BOOST que possibilita a utilização da potência máxima de +20 dBm, que por sua vez foi definida.

Após realizadas as configurações necessárias para o correto funcionamento do módulo LoRa, foram desenvolvidas as funções responsáveis pela transmissão e receção de dados. Os dois transceptores utilizam ambas as funções uma vez que o sistema de comunicação LoRa é bidirecional. É importante referir que estas funções tiveram por base os diagramas fornecidos pelo *datasheet*, semelhantes aos já apresentados nas secções 3.5.3.4 e 3.5.3.5, existindo no entanto algumas alterações de modo a adaptá-las à solução pretendida.

5.1.3 Processo de Transmissão de Dados

O fluxograma referente ao processo de transmissão de dados através do módulo de rádio LoRa encontra-se apresentado na Figura 5.3. Nesta função, numa primeira fase, são realizadas algumas preparações de modo a permitir o correto envio dos dados. Inicialmente o módulo de rádio é colocado em modo *standby* para permitir aceder aos registos de configuração e ao *buffer* de dados FIFO. A alteração entre os modos de funcionamento do módulo de rádio é sempre realizado através da função **Op_Mode()**. De seguida, é preparado o apontador da memória FIFO, que indicará a posição na memória onde serão colocado os dados que se pretende enviar. Posteriormente é chamada a função **Switch_Tx()**, que coloca os pinos do módulo de rádio LoRa em modo de transmissão, e são configuradas as interrupções. Previamente ao envio é ainda chamada a função **doConversion()**, que simula a leitura dos dados do sensor, neste caso do potenciómetro, e que seguidamente são carregados para a memória FIFO. Numa segunda fase, o módulo de rádio é colocado no modo de transmissão e, no final do envio, é chamada a função **Switch_RX()** que coloca os pinos do módulo rádio LoRa no modo de receção.

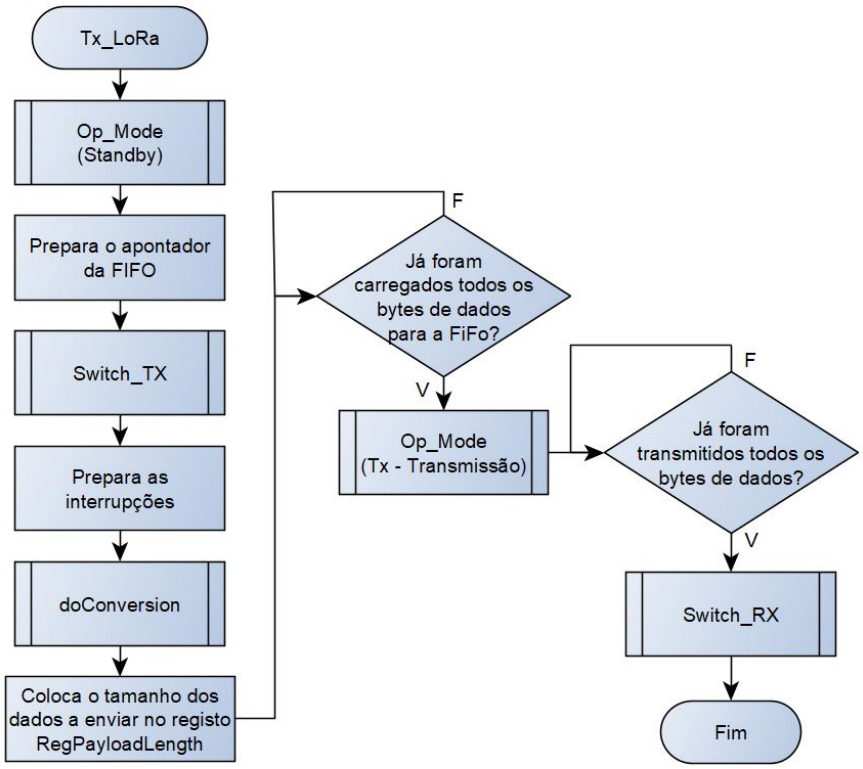


Figura 5.3: Fluxograma da função de transmissão, Tx_LoRa().

5.1.4 Processo de Receção de Dados

Ao contrário do que acontece na transmissão de dados, a receção é efetuada de maneira diferente nos dois transceptores. Isto deve-se ao objetivo definido para cada um destes e sobretudo ao tipo de alimentação que cada um apresenta. O fluxograma da Figura 5.4 apresenta a função desenvolvida para a receção de dados no *end device*, que tem por base o modo de funcionamento Rx_Single disponibilizado pelo módulo de rádio LoRa. Foi optado por este modo de funcionamento uma vez que este transceptor possui uma alimentação externa e, de modo a aumentar a duração da mesma, este apenas operará durante curtos períodos de tempo em modo de receção, sendo posteriormente colocado em modo *sleep*. Por outro lado, como a *gateway* possui uma alimentação contínua, esta terá por base o modo de funcionamento Rx_Continuous. A Figura 5.5 representa a função desenvolvida para receção de dados na *gateway*.

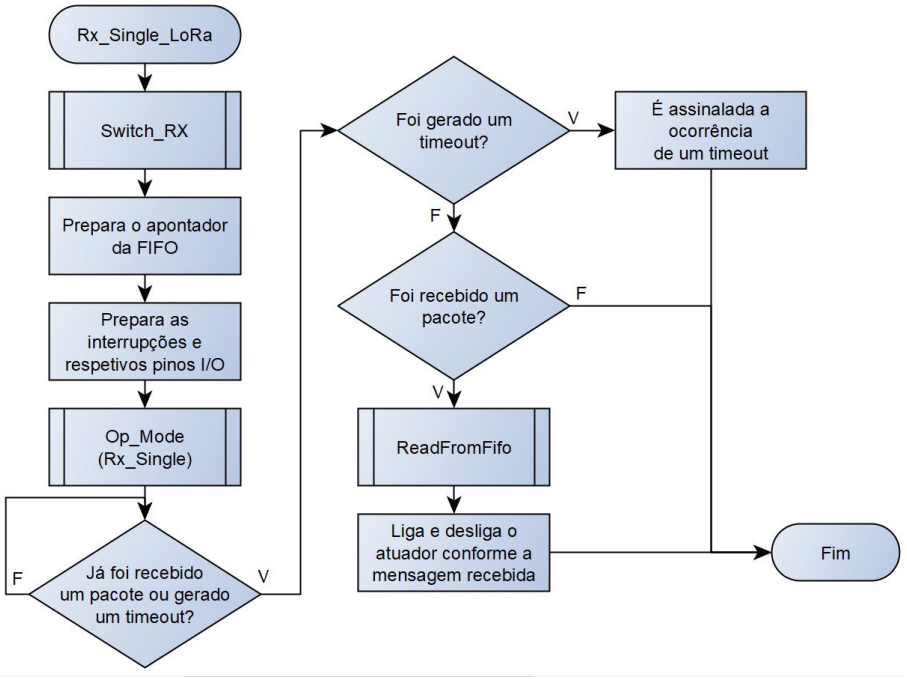


Figura 5.4: Fluxograma da função de receção, Rx_Single_LoRa().

A função **Rx_Single_LoRa()** inicia com colocação dos pinos do módulo de rádio LoRa no modo de receção através da função **Switch_Rx()**, seguido da preparação do apontador para a memória FIFO e das interrupções necessárias, nomeadamente a **RxDone** e **RxTimeout**. Posteriormente o módulo é colocado no modo de receção (Rx_Single) e é esperada a ocorrência de uma interrupção, nomeadamente a receção de um pacote de dados, **RxDone**, ou a ocorrência de um *timeout*, **RxTimeout**. Caso ocorra um *timeout* este é assinalado, caso seja recebido um pacote de dados, é efetuada a sua respetiva leitura da memória FIFO através da função **ReadFomFifo()** e é ligado ou desligado o atuador consoante o conteúdo da mesma.

A função **ReadFromFifo()** é partilhada por ambas as funções de receção desenvolvidas e tem como objetivo retirar a informação recebida da memória FIFO para que posteriormente possa ser utilizada. Inicialmente é colocado o apontador da FIFO no endereço da última mensagem recebida (linha 2). Seguidamente, esta é retirada através do registo REG_FIFO, *byte* a *byte*, consoante o tamanho da mensagem recebida, indicada pelo registo *RegRxNbBytes* (linha 3 e 4). Por último, é desativada a *flag* **Rx_Done** que indicava a receção de uma nova mensagem (linha 5). O excerto de código seguinte diz respeito à função **ReadFromFifo()** desenvolvida:

```
1  void ReadFromFifo(char *payload){
     // Prepare pointer
2    LoRa_WriteRegister(REG_FIFO_ADDR_PTR,
            (LoRa_ReadRegister(REG_FIFO_RX_CURRENT_ADDR)));

     // Retrieve data
3    for (int i = 0; i < LoRa_ReadRegister(REG_RX_NB_BYTES); i ++){
4     payload[i]= LoRa_ReadRegister(REG_FIFO);
     }

     // Disable RxDone flag
5    LoRa_WriteRegister(REG_IRQ_FLAGS,
          (LoRa_ReadRegister(REG_IRQ_FLAGS) | RX_DONE_FLAG_MASK));
}
```

Além da utilização deste modo de receção, Rx_Single, o módulo de rádio LoRa disponibiliza seis pinos I/O que permitem indicar a ocorrência de diversos eventos durante o processo de transmissão e receção, consoante a sua configuração. Isto permite evitar o *polling* periódico do registo *RegIrqFlags* que indica a ocorrência das interrupções. Com a utilização destes pinos I/O apenas é necessário verificar o estado dos pinos do lado do microcontrolador que correspondem às interrupções pretendidas. Deste modo é possível reduzir o consumo energético por parte do módulo de rádio. No *end device* foram utilizados dois pinos I/O, DIO0 e DIO1, com as respetivas interrupções associadas *RxDone* e *RxTimeout*.

A função **Rx_Single_LoRa()** apenas fornece um determinado período de tempo para receção de um pacote de dados. Após este tempo, e caso não seja recebido nenhum pacote, é gerada a interrupção **RxTimeout**. O dimensionamento do tempo para a ocorrência de um *timeout* é calculado a partir da equação 3.3 já apresentada na secção 3.5.3.5. Neste caso foi definido um tempo de *timeout* de 3 segundos, que para uma BW de 125 kHz e um SF de 11 resulta num *SymbTimeout* de 183.

A função **Rx_Continuous_LoRa()** é utilizada na *gateway*. Esta possui um principio de funcionamento mais simples, uma vez que apenas fica continuamente à espera de dados. Assim como a função anterior, esta inicia com a utilização da função **Switch_Rx()** e com a preparação do apontador para a memória FIFO e das interrupções. Posteriormente, o módulo é colocado no modo de receção (Rx_Continuous) e este fica indeterminadamente à espera que seja recebida uma mensagem. Após a receção, esta é retirada da memória de dados FIFO e o módulo de rádio é colocado em modo *standby*. Uma vez que o consumo energético neste módulo não é o factor mais importante, a verificação da ocorrência de

uma interrupção devido à receção de um pacote é realizada através do *polling* do registo *RegIrqFlags*, diminuindo assim o número de ligações entre o micro-controlador e o módulo de rádio. Esta função será ligeiramente modificada com a introdução do protocolo MQTT no capítulo 5.2.3, no entanto este é o princípio de funcionamento base.

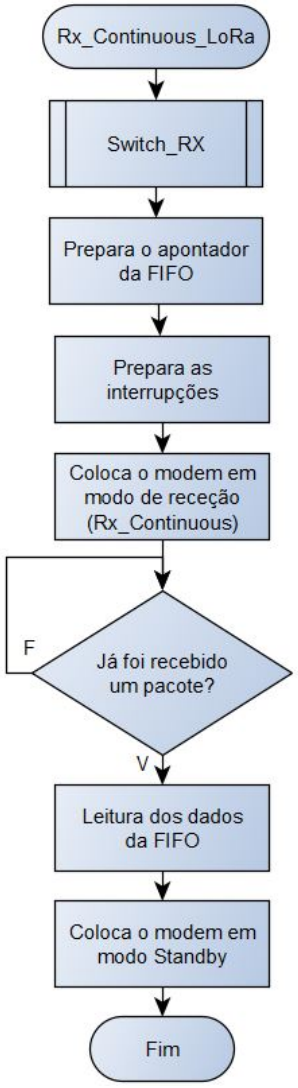


Figura 5.5: Fluxograma da função de receção, Rx_Continuous_LoRa().

5.1.5 Princípio de Funcionamento do *End Device* e da *Gateway*

O princípio de funcionamento do *end device* tem por base as funções transmissão e receção de dados, assim como a de configuração do módulo de rádio LoRa abordadas nas secções anteriores, nomeadamente a função **Tx_LoRa()**, **Rx_Single_LoRa()** e **LoRa_Config()**. O *end device* inicia com a configuração dos diversos periféricos necessários, especificamente os pinos I/O, SPI e ADC, seguido da configuração do módulo de rádio LoRa. Posteriormente dá-se início ao ciclo principal que inicia com a transmissão de uma mensagem. Após a transmissão, este é colocado em modo de receção onde escuta o meio por uma mensagem durante três segundos. Este é o único período de tempo no qual o *end device* poderá receber uma mensagem. No final o módulo de rádio é colocado no modo *Sleep* durante 15 segundos de modo a diminuir o seu consumo energético. Após isto, todo o processo é repetido. Na Figura 5.6 é apresentado o fluxograma referente ao princípio de funcionamento do *end device*. A utilização destes tempos resulta num *duty cycle* do *end device* de aproximadamente 19 % o que benefícia eficiência energética do sistema. Por outro lado o tempo de resposta do sistema é sensivelmente de 18 segundos. Estes valores foram atribuídos arbitrariamente porém, uma análise mais detalhada destes será efetuada na secção 6.3.

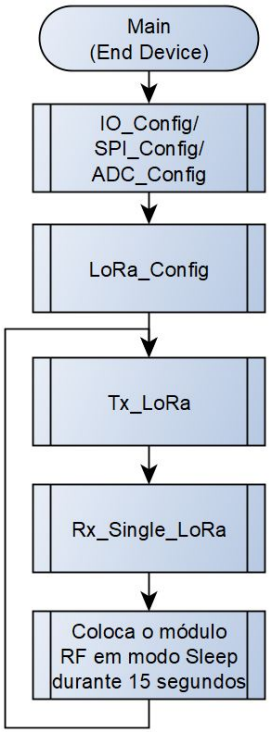


Figura 5.6: Fluxograma referente ao modo de funcionamento do *end device*.

Na Figura 5.7 é apresentado o fluxograma referente ao princípio de funcionamento da *gateway*. No que diz respeito ao funcionamento geral, esta inicia de forma semelhante ao *end device*, numa primeira fase são configurados os diversos periféricos necessários, nomeadamente os pinos I/O e o SPI, seguido da configuração do módulo de rádio LoRa. Uma vez que este transceptor funciona como um nó intermediário entre o meio físico e a *Internet*, este necessita de estabelecer uma comunicação com a *Internet* através de uma ligação Wi-Fi. O estabelecimento da ligação Wi-Fi é realizada através da função **Init_WiFi()**. Por sua vez, a função **Init_MQTT()** efetua as inicializações necessárias para possibilitar a troca de mensagens com o servidor através do protocolo MQTT.

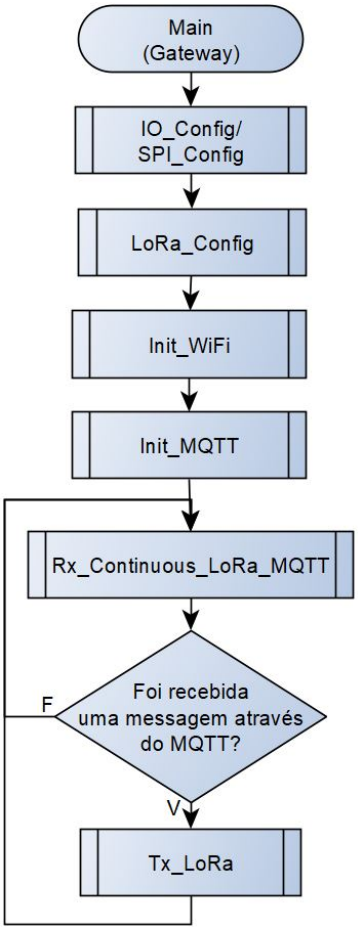


Figura 5.7: Fluxograma referente ao modo de funcionamento da *gateway*.

Após efetuadas todas as inicializações, o ciclo principal da *gateway* inicia com a função **Rx_Continuous_LoRa_MQTT()**, que permite a escuta contínua do meio por uma mensagem, não só vinda do *end device*, mas também do servidor. Após a receção de uma mensagem proveniente do *end device*, é verificado se

foi recebida alguma mensagem por parte do servidor, caso esta situação ocorra a *gateway* reencaminha essa mensagem para o *end device*, através de função **Tx_-LoRa()**, e no final do envio retorna ao inicio do ciclo, para a função **Rx_Continuous_LoRa_MQTT()**. Caso não seja recebida nenhuma mensagem por parte do servidor, o programa retorna de imediato ao início do ciclo.

As funções **Init_WiFi()** e **Init_MQTT()**, assim como a **Rx_Continuous_LoRa _MQTT()** serão abordadas com maior detalhe na secção 5.2, no entanto a referência destas neste capítulo é fundamental para a sua compreensão.

A Figura 5.8 pretende resumir a interação entre os dois transceptores através de um diagrama temporal. Em forma de resumo, o *end device* possui um ciclo fixo no qual inicia sempre com a transmissão de um pacote, seguido de uma janela com um período de 3 segundos para a receção de uma mensagem proveniente da *gateway*, sendo este o único período disponível durante o ciclo para efetuar esta operação. Para terminar o ciclo, o *end device* é colocado em modo *sleep* durante 15 segundos de modo a maximizar a duração da bateria. Por outro lado a *gateway* encontra-se continuamente a escutar o meio por mensagens provenientes do *end device* e do servidor. Porém, caso seja recebida uma mensagem proveniente do servidor, esta só será enviada após a receção de uma mensagem por parte do *end device*, uma vez que a janela de receção deste só abrirá momentaneamente após uma transmissão. No final da transmissão, por parte da *gateway*, este retorna ao modo de receção. No caso de não ser recebida nenhuma mensagem do servidor, a *gateway* retorna imediatamente ao modo receção após a chegada de uma mensagem do *end device*.

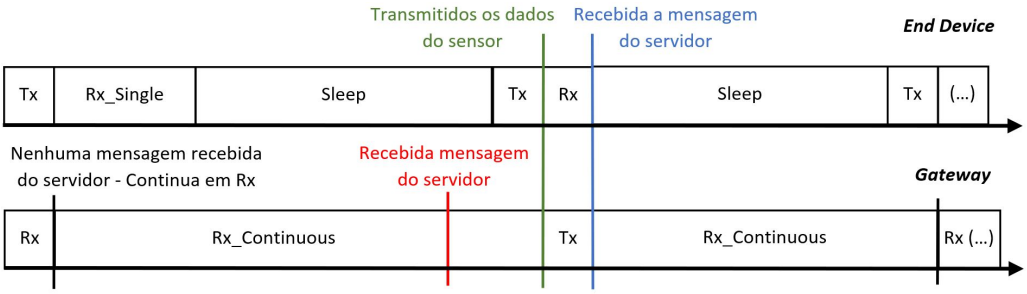


Figura 5.8: Diagrama de funcionamento do sistema de comunicação LoRa.

5.2 Sistema de Comunicação com o *Back-End*

Ao nível de *hardware* o sistema de comunicação com o *back-end* apenas é constituído pela *gateway* já abordada na secção anterior, secção 5.1. No que diz respeito ao *firmware* desenvolvido este será dividido em duas partes, a primeira será referente ao *firmware* desenvolvido do lado da *gateway* e a segunda referente ao lado do servidor.

5.2.1 Estabelecimento de uma Ligação Wi-Fi na *Gateway*

Para o estabelecimento de uma comunicação através do protocolo MQTT é necessário primeiramente iniciar uma conexão a uma rede Wi-Fi. Para efetuar esta conexão foi utilizada a biblioteca **ESP8266WIFI.h**. Esta biblioteca em C++ foi desenvolvida para a família de produtos baseados no ESP8266 de modo a facilitar a conexão dos mesmos à *internet*. A função **init_WiFi()** foi desenvolvida com recurso a esta biblioteca para estabelecer uma comunicação Wi-Fi. As seguintes linhas de código são essenciais para o correto estabelecimento de uma ligação:

```
1  void init_WiFi(){
2    WiFi.begin(ssid, password);
3    while (WiFi.status() != WL_CONNECTED) {
4      delay(500);
5      Serial.print(".");
     }
   }
```

A função **WiFi.begin()** inicia o estabelecimento da comunicação à rede Wi-Fi pretendida através da indicação do nome da rede e da *password*. Para isso são usados os respetivos parâmetros **ssid** e **password** (linha 2). A função **WiFi.status()** presente no ciclo **while** permite verificar o estado da ligação (linha 3). O programa apenas correrá caso a conexão seja estabelecida com sucesso.

5.2.2 Implementação do Protocolo MQTT na *Gateway*

Assegurada uma ligação Wi-Fi é necessário estabelecer uma comunicação com o servidor através do protocolo MQTT. Para efetuar esta ligação foi utilizada a biblioteca **PubSubClient.h**. Esta biblioteca facilita a implementação de um cliente simples para a troca de mensagens com um servidor que suporta MQTT.

Numa primeira fase é necessário criar um cliente. Neste caso, é efetuada um criação parcial onde apenas é indicado o nome do cliente (linha 1 e 2).

Posteriormente é configurada a ligação com o servidor através da função **client.setServer()**, onde é necessário indicar o IP do servidor e a porta utilizada (linha 3). Além disso é definida a função *Callback*, isto é, a função que é chamada sempre que é recebida uma mensagem num dos tópicos subscritos. Esta é definida através da função *client.setCallback* que recebe como parâmetro um apontador para a respetiva função, neste caso designada de **callback()** (linha 4).

```
1  WiFiClient espClient;
2  PubSubClient client(espClient);

   /* MQTT init */
3  client.setServer(mqtt_server, port);
4  client.setCallback(callback);
```

Como já foi referido, a função **callback()** é chamada sempre que é recebida uma nova mensagem num dos tópicos subscritos. Esta recebe como parâmetros o tópico no qual foi recebida a mensagem, o conteúdo e o tamanho da mesma (linha 1). Nesta situação em particular a mensagem recebida é armazenada num vetor, **MQTT_Message[]**, para posteriormente ser reencaminhada para o *end device* (linha 2 e 3). Uma vez retirada a mensagem é ativada uma *flag*, nomeadamente *flag_MQTT_Message_Received*, que indica a receção de uma nova mensagem (linha 4). O excerto de código referente à função **callback()** encontra-se apresentado abaixo:

```
1  void callback(char* topic, byte* payload, unsigned int length) {
2    for (int i = 0; i < length; i++) {
3      MQTT_Message[i] = payload[i];
     }
4    flag_MQTT_Message_Received = 1;
   }
```

É importante referir que neste trabalho foram criados dois tópicos. Um dos tópicos é referente aos dados do sensor, que circulam no sentido da *gateway* para o servidor, designado de **dataFromESP8266**. O outro tópico diz respeito ao estado do atuador, designado de **LedStatus** e neste os dados circulam no sentido do servidor para a *gateway*.

Depois de configurada a ligação ao servidor e desenvolvida a função **callback()** é estabelecida a conexão ao *broker* (servidor). Para isto apenas é necessário utilizar a função *client.connect(client_ID)* disponibilizada pela biblioteca **ESP8266WIFI.h**, onde o **client_ID** é o nome atribuído ao cliente.

5.2.3 Processo de Receção de Dados na *Gateway*

Como foi referido na secção 5.1.4, a adição do protocolo MQTT provocou alterações no funcionamento da *gateway*, nomeadamente na função de receção de dados, denominada agora de **Rx_Continuous_LoRa_MQTT()**. O fluxograma final desta função encontra-se apresentado na Figura 5.9.

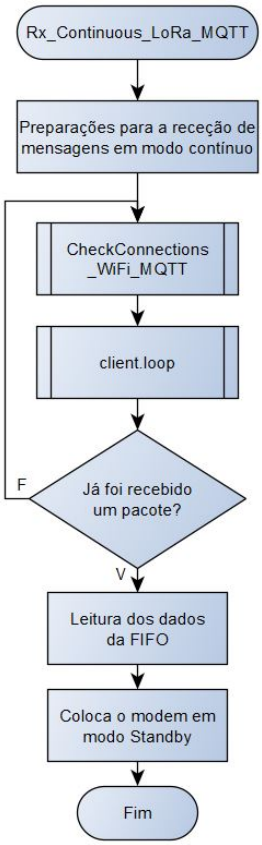


Figura 5.9: Fluxograma da função de receção, Rx_Continuous_LoRa_MQTT().

De uma forma geral a estrutura da função manteve-se em relação há abordada na secção 5.1. A configurações e preparações necessárias para a receção de dados mantiveram-se, no entanto, durante a espera por um novo pacote foram acrescentadas duas novas funções, nomeadamente **CheckConnections_WiFi_MQTT()** e **client.loop()**. Após a receção de um pacote, a função manteve o principio de funcionamento anterior. A função **CheckConnections_WiFi _MQTT()** tem como objetivo verificar o estado da ligação ao servidor via MQTT (linha 2) e à rede Wi-Fi (linha 4). Caso alguma destas ligações seja interrompida, é iniciada uma tentativa de reconexão da respetiva ligação perdida através das respetivas funções, **reconnect()** e **reconnectWiFi()**. O código referente à função **CheckConnections_WiFi_MQTT()** é apresentado de seguida:

```
1  void CheckConnections_WiFi_MQTT() {
2    if (!client.connected()) {
3        reconnect();
     }

4    if (WiFi.status() != WL_CONNECTED){
5       reconectWiFi();
    }
  }
```

A função **reconnect()** pretende reestabelecer a comunicação com o *broker*. A conexão é sempre efetuda através da função **client.connect()** (linha 3). Após a conexão é necessário subscrever nos tópicos nos quais se pretende receber mensagens. Para esse efeito é usada a função **client.subscribe("topic")** (linha 4), que recebe como parâmetro o tópico ao qual se pretende subscrever. O excerto de código seguinte diz respeito à função **reconnect()**:

```
1  void reconnect() {
2     while (!client.connected()) {
3        if (client.connect("ESP8266Client")) {
4            client.subscribe(TOPIC_SUBSCRIBE);
5        } else {
               // Wait 3 seconds before retrying
6              delay(3000);
         }
     }
   }
```

No que diz respeito à função **reconnectWiFi()**, esta tem como objetivo reestabelecer a ligação à rede Wi-Fi. O seu princípio de funcionamento é semelhante à função **init_WiFi()** já apresentada na secção 5.2.1.

Relativamente à função **client.loop()**, esta é intrínseca à biblioteca **PubSubClient.h** e deve ser chamada regularmente para permitir que o cliente processe as mensagens recebidas do servidor e mantenha a conexão com o mesmo. Devido a estas caraterísticas e uma vez que o programa da *gateway* passará grande parte do seu tempo no ciclo de verificação de receção de um pacote LoRa, optou-se pela colocação desta função nesta zona do código.

Por último, e de modo a reencaminhar as mensagens provenientes do *end device*, foi necessário modificar a função **ReadFromFifo()** da *gateway*, através da adição da seguinte linha de código:

```
client.publish(TOPIC_PUBLISH, transmit_buffer);
```

De forma sucinta, a função **client.publish()** publica a mensagem presente na variável **transmit_buffer** no tópico **TOPIC_PUBLISH** e envia esta para o *broker*.

5.2.4 Implementação do Protocolo MQTT no *Back-End*

O servidor é um componente fundamental deste trabalho, uma vez que é responsável por desempenhar duas tarefas essenciais, primeiro desempenha o papel de *broker* e de pseudo-cliente, e segundo é responsável por responder ao pedidos provenientes do *front-end*. Este segundo ponto será abordado na secção 5.3.

Para desempenhar o papel de *broker* foi utilizado o Mosca uma vez que permite ser utilizado com o NodeJS. Para facilitar a sua implementação é disponibilizada uma biblioteca denominada de **mosca** (linha 1). Nesta situação, para a inicialização do *broker* apenas foi indicada a porta que este deverá utilizar, neste caso a porta "1883" (linha 2). Para a inicialização do servidor é utilizada função **broker.on()** (linha 4), que irá iniciar este através da variável **broker** originada de uma nova instância de **mosca.Server()** (linha 3) criada a partir do objeto *settings* que possui a porta a utilizar como atributo. O código para a criação e inicialização do *broker* é apresentado de seguida:

```
   // MQTT broker
1  var mosca= require('mosca')
2  var settings = {port: 1883}
3  var broker = new mosca.Server(settings)

   // Init broker/server
4  broker.on('ready', ()=>{
5      console.log('Broker is ready!')
   })
```

Simultaneamente à criação do *broker* é também criado um pseudo-cliente no próprio servidor. Este tem como objetivo receber as mensagens provenientes da *gateway* e colocá-las na base de dados. Numa primeira fase é inicializado o cliente com recurso à biblioteca **mqtt** (linha 1). Posteriormente é criada uma nova instância de **mqtt.connect()**, que tem como parâmetro o endereço e a porta do servidor, e que pode ser acedida através da variável **client** (linha 2). De seguida, através das funções **client.on()** e **client.subscribe()** é, respetivamente, inicializado o cliente e é subscrito o tópico no qual este cliente pretende receber mensagens. O código referente à criação e inicialização do cliente encontra-se apresentado abaixo:

```
   // MQTT client
1  var mqtt = require('mqtt')
2  var client = mqtt.connect('mqtt://localhost:1883')
3  var topic2 = 'dataFromESP8266'

   // Init client
4  client.on('connect', () =>{
5      console.log('Client is ready!');
6      client.subscribe(topic2)
   })
```

5.2.5 Conexão entre o *Back-End* e MongoDB

De forma semelhante é ainda estabelecida uma comunicação com a base de dados, MongoDB, para que posteriormente possam ser armazenados os dados recebidos. Inicialmente é carregada a biblioteca **mongoose** que facilita a interação entre o servidor e a base de dados MongoDB (linha 1). Para o estabelecimento e comunicação é utilizada a função **mongoose.connect()** que necessita de um parâmetro fundamental, o *Uniform Resource Identifier* (URI) que é composto pelo utilizador e a *password*, e pelo endereço da base de dados (linha 3). Neste deve ser especificada a porta a utilizar seguido do nome da base de dados. De forma a estabelecer uma comunicação contínua entre o servidor e a base de dados é utilizada a função **connection.once()** (linha 5).

```
   // MongoDB with mongoose
1  var mongoose = require('mongoose');
2  var uri = 'mongodb://root:1234@localhost:27017/appDB'

   // Connection to MongoDB
3  mongoose.connect(uri, { useNewUrlParser: true,
           useCreateIndex: true});
3  const connection = mongoose.connection;
5  connection.once('open', () => {
6     console.log("MongoDB database connection
                established successfully!");
    })
```

Após estabelecida a comunicação com a base de dados, a transferência de informação entre esta e o servidor é possível. Com foi referido anteriormente, a criação do pseudo-cliente no servidor tem como objetivo receber a informação de um determinado tópico proveniente do *end device* e colocá-la na base de

dados. O excerto de código seguinte demonstra como é efetuado o armazenamento. Numa primeira fase é colocada na variável myCol uma nova instância de **mongoose.connection.collection()** onde é indicada a localização para o armazemento dos dados, isto é, onde é definida a coleção dentro da base de dados que se pretende utilizar, neste caso **datas** (linha 1). Posteriormente, os dados são inseridos através função myCol.insertOne() (linha 2), que envia a informação num formato semelhante ao JSON (linha 3, 4, 5 e 6).

```
1  var myCol = mongoose.connection.collection('datas')
2  myCol.insertOne({
3    data: message,
4    createdAt: new Date,
5    updatedAt: new Date,
6    __v: 0
7    }, ()=>{
8          console.log('Data saved to MongoDB!', message)
     })
```

5.3 Ligação entre *Back-End* e *Front-End*

Para a visualização da informação obtida do sensor e efetuar o controlo sobre o atuador foi desenvolvido um UI com recurso ao ReactJS.

5.3.1 Criação do *Front-End*

Para facilitar a instalação desta biblioteca o NodeJS disponibiliza uma ferramenta denominada de *Node Package Manager* (npm) que, como o nome indica, permite gerir pacotes/bibliotecas no NodeJS. Para a instalação e criação de uma aplicação em ReactJS é necessário executar os seguintes comandos:

```
npm install npx
npx create-react-app iot-app
```

Após a execução destes comandos todos os ficheiros base para o desenvolvimento do *front-end* são criados.

5.3.2 Criação do *Back-End*

Além do *front-end* é necessário criar o *back-end* para que este possa comunicar com este e o restante sistema. Para isso foi criada uma nova pasta, denominada de ***back-end***, e dentro desta foi efetuado o seguinte comando de modo a criar os ficheiros base para o desenvolvimento do *back-end*:

```
npm init
```

Seguidamente foi ainda criado o ficheiro ***server.js*** que contém todo o código referente ao *back-end*. Algumas partes do código presentes neste ficheiro já foram abordadas nas secções 5.2.4 e 5.2.5. Ainda no ***back-end*** foram criadas duas pastas, ***models*** e ***routes***. A pasta ***models*** contém os ficheiros que definem a estrutura de como serão armazenados ou procurados os dados na base de dados e a pasta ***routes*** contém os ficheiros responsáveis por responder aos diferentes pedidos provenientes do *front-end*.

A Figura 5.10 representa a organização dos ficheiros referentes ao *front-end* e *back-end* de modo a facilitar a sua compreensão.

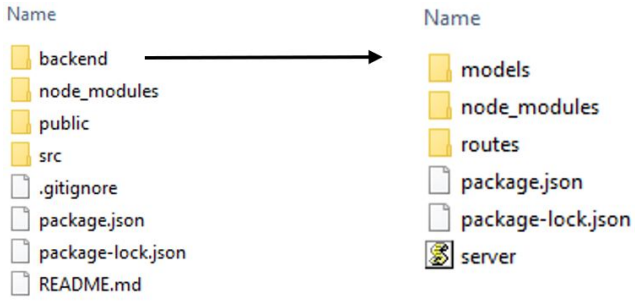


Figura 5.10: Estrutura dos ficheiros relativos ao *back-end* e *front-end*.

5.3.3 Desenvolvimento do *Back-End*

O ficheiro ***server.js*** contém grande parte do código referente ao *back-end*. Este encontra-se divido em duas componentes. A primeira, já abordada, diz respeito ao protocolo MQTT e à conexão com a base de dados. A segunda é relativa à ligação com o *front-end*. De forma a responder aos diferentes pedidos provenientes do *front-end* foi utilizada biblioteca **express** (linha 1). O excerto de código seguinte é referente à inicialização do servidor através desta biblioteca. Para isto foi criada um instância de um objeto **espress** denominada de **App** (linha 2). A partir desta é inicializado o servidor que fica à escuta do meio, através a função **app.listen()** (linha 9), na porta 5000 definida pela variável **port** (linha 3). A função **app.use(express.json())** indica que todos os pedidos irão ser interpretados num formato JSON (linha 6). Além disso foram ainda definidos dois ficheiros responsáveis por responder aos diferentes pedidos efetuados nos *routes* "/data" e "/status" (linha 7 e 8). Por exemplo, no caso do *route* "/data" foi utilizada a função **app.use('/data', dataRouter)**.

```
1  var express = require('express');

2  const app = express();
3  const port = process.env.PORT || 5000;
4  const dataRouter = require('./routes/data');
5  const statusRouter = require('./routes/status');

6  app.use(express.json());
7  app.use('/data', dataRouter);
8  app.use('/status', statusRouter);

9  app.listen(port, () => {
10      console.log('Server is running on port:' + port);
   })
```

No que diz respeito aos ficheiros presentes no pasta ***routes*** estes podem ser referidos como *middleware*. Neste caso foram criados dois ficheiros, um responsável pelos pedidos referentes às leituras do sensor, denominado de ***data.js***, e o outro é responsável pelos pedidos relativos ao atuador, desginado de ***status.js***. Os ficheiros são semelhantes e deste modo apenas será abordado o último. O excerto de código seguinte é referente ao ficheiro ***status.js***. Este inicia com inicialização da variável **router** vinculada a uma instância de **require('express').Router()** (linha 1). A partir desta variável são chamadas todas a funções de roteamento. Neste caso temos a função **router.route('/').get((req, res) => {})** (linha 3) que, após um pedido **GET** no *route* **'http://localhost:5000/'**, irá executar uma determinado função. Nesta situação é executada a função **Status.find().sort('-_id').limit(12)** que irá procurar na base de dados os doze valores mais recentes, através do parâmetro **"id"**, com uma estrutura igual à definida pela variável **Status**, sendo posteriormente retornados ao *front-end* (linha 4 e 5). Os ficheiros referentes à estrutura dos documentos serão abordados mais à frente.

```
1  const router = require('express').Router();
2  let Status = require('../models/status.model');

3  router.route('/').get((req, res) => {
4    Status.find().sort('-_id').limit(12)
5    .then(status => res.json(status))
6    .catch(err => res.status(400).json('Error: ' + err));
   });
```

De forma semelhante, este ficheiro possui ainda a função **router.route('/add').post((req, res) =>** {}) (linha 1) que, após um pedido **POST**

no *route* **'http://localhost:5000/add'**, irá armazenar na base de dados, através da função **newStatus.save()** (linha 4), o conteúdo da mensagem proveniente do *front-end*. É neste *route* que são efetuados os diferentes pedidos que permitem ligar ou desligar o atuador. Desta forma, no final da execução do código referente ao roteamento é acrescentada a função **client.publish(topic, status)** que irá transmitir de imediato o estado do atuador para a *gateway* (linha 7).

```
1  router.route('/add').post((req, res) => {
2     const status = req.body.status;
3     const newStatus = new Status({status});

4     newStatus.save()
5       .then(() => res.json('Data added!'))
6       .catch(err => res.status(400).json('Error: ' + err));

7     client.publish(topic, status)
   });
```

Relativamente aos ficheiros presentes a pasta ***models***, estes definem a estrutura dos documentos armazenados na base de dados. Nesta aplicação foram criados dois ficheiros, um com a estrutura para os dados obtidos do sensor e outra para o estado do atuador, designados respetivamente de ***data.model.js*** e ***status.model.js***. Para a criação da estrutura é inicialmente criada a variável **Schema** vinculada a uma instância de **mongoose.Schema** (linha 2). A partir desta variável é criada a estrutura, no formato JSON, onde são indicados os parâmetros e outras propriedades opcionais (linha 3). Este exemplo é simples onde apenas é adicionado um parâmetro denominado de **status** (linha 4). Adicionalmente é ainda colocado o parâmetro **timestamp** que acrescenta a data e hora respetiva ao momento de armazenamento (linha 5). Além destes, a base de dados irá gerar um outro parâmetro automaticamente, um identificador único para cada documento criado. Este parâmetro é denominado de "**_id**" e é utilizado na organização das tabelas. O código presente no ficheiro ***status.model.js*** encontra-se apresentado abaixo. O desenvolvimento do ficheiro ***data.model.js*** é semelhante a este.

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3  const statusSchema = new Schema ({
4      status : { type: String, require: true},
   }, {
5      timestamps: true,
   });
```

5.3.4 Desenvolvimento do *Front-End*

Para a apresentação dos dados foi desenvolvida uma aplicação simples em ReactJS. Esta biblioteca disponibiliza um conjunto de ficheiros base para a elaboração do *front-end*, porém foi necessário editar, assim como acrescentar alguns ficheiros de modo a obter-se o aspeto e funcionalidades pretendidas. Todos estes ficheiros situam-se no diretório ***src***.

A estrutura principal da aplicação é definida no ficheiro ***App.js***. Esta aplicação é constituída por três componentes, nomeadamente uma barra de navegação, e duas páginas, uma para a exibição dos dados do sensor armazenados na base de dados e outra para a exibição do estado do atuador e efetuar o controlo do mesmo. Para cada um destes componentes foi atribuído um ficheiro designados, respetivamente, por ***navbar.component.js***, ***actuator.component.js*** e ***datalist.component.js***. O código desenvolvido referente a cada um destes componentes encontra-se dentro do diretório ***components***. A barra de navegação assim como os restantes componentes foram desenvolvidos com recurso à biblioteca *bootstrap* de modo a facilitar a sua implementação.

A barra de navegação é o componente mais simples e de menor importância. Devido a isto é atribuída maior ênfase aos restantes componentes. Dentro destes, o envio e a recepção da informação através do *front-end* é o aspeto fundamental, tornando-se o foco principal desta secção. Por sua vez o *web design* destes componentes não será abordado.

Para a receção de dados foi desenvolvida a função **componentDidMount()** (linha 1) que, por sua vez, utiliza o método **GET**. De forma a facilitar a realização dos pedidos HTTP foi utilizada a biblioteca **axios**. Para a realização de um pedido **GET** é utilizada a função **axios.get('route')** recebendo como parâmetro o *route* no qual se pretende fazer o pedido (linha 2). Os dados recebidos serão posteriormente exibidos numa tabela. A função **componentDidMount()**, presente no ficheiro ***actuator.components.js***, encontra-se apresentada abaixo:

```
1  componentDidMount() {
2    axios.get('http://localhost:5000/status/')
3    .then(response => {
4       this.setState({ status: response.data})
     })
5    .catch((error) => {
6       console.log(error);
     })
   }
```

A função **componentDidMount()** é, de igual forma, utilizada para a obtenção dos dados do sensor armazenados na base de dados para que posteriormente também possam ser exibidos numa outra tabela, porém o *route* utilizado na função **axios.get('route')** é diferente, mais precisamente **'http://localhost:5000/data/'**. De forma a que os dados do sensor presentes no respetivo separador do *front-end* se encontrem atualizados com os dados presentes na base de dados sem a necessidade de o utilizador atualizar a página de forma manual, foi implementada uma função que efetua uma atualização deste separador a uma determinada frequência. Deste modo é mantido um certo nível de sincronismo entre o *front-end* e a base de dados.

No componente referente ao atuador pretende-se também efetuar o controlo do mesmo. Para isso são utilizados dois botões, um para ligar e outro para desligar, sendo associado a cada um destes uma função. A função utilizada para ligar o atuador, denominada de **sendOn()**, encontra-se apresentada no excerto de código seguinte. Esta diferencia-se da função anterior, **componentDidMount()**, pela utilização do método **POST**, através da função **axios.post('route', status)** (linha 4). Por sua vez, este método necessita de um parâmetro adicional que contêm a informação que se pretende enviar. Neste caso é enviada a mensagem "ON" através da variável **status** (linha 2 e 3). No final da execução da função é atualizada a página de modo a colocar de imediato este pedido de alteração do estado do atuador na respetiva tabela (linha 6). A função desenvolvida para desligar o atuador é semelhante a esta, porém é alterado o conteúdo da mensagem a enviar para "OFF".

```
1  async sendON() {
2     const status = {
3       status: "ON"
       }

4     axios.post('http://localhost:5000/status/add', status)
5       .then(response =>  console.log(response.data));

6     window.location = '/actuator';
   }
```

Em forma de resumo, o esquema apresentado na Figura 5.11 pretende demonstrar as interações existentes entre o *front-end* e o *back-end*, iniciadas pelos diferentes pedidos HTTP presentes na aplicação.

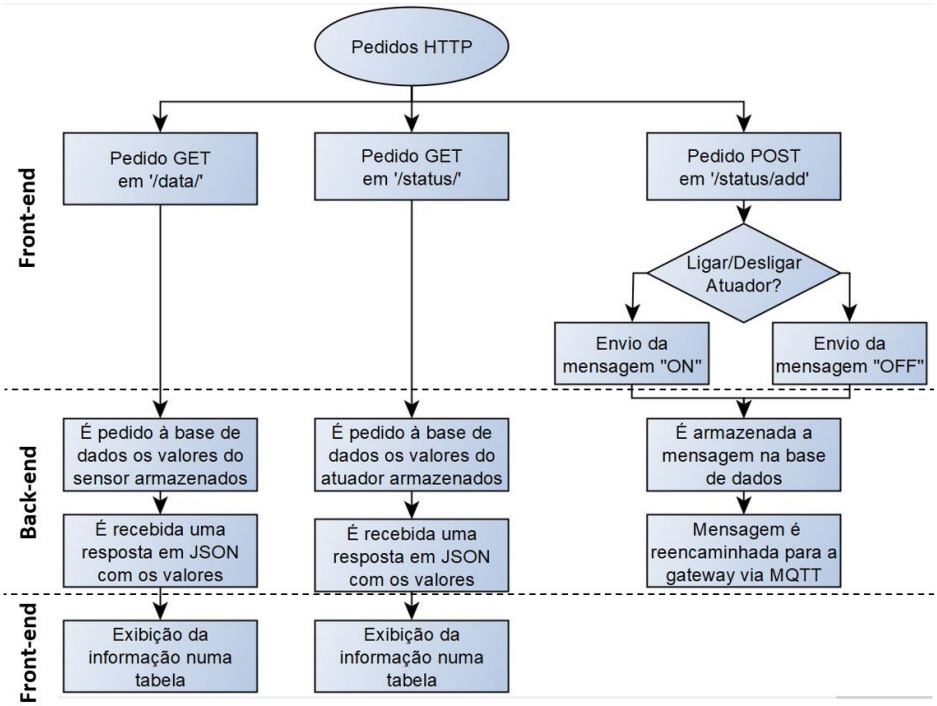


Figura 5.11: Esquema com os diferentes pedidos HTTP presentes na aplicação.

Capítulo 6

Testes e Resultados

Neste capítulo são apresentados os resultados obtidos através do sistema desenvolvido. Serão abordados os resultados obtidos ao nível da aplicação e ao nível do alcance físico do sistema. Além disso, é realizada uma análise ao *duty cycle* e tempo de resposta do sistema de comunicação LoRa.

6.1 Resultados

De forma a facilitar a exposição dos resultados obtidos estes serão divididos em dois grupos. Numa primeira fase são apresentados os resultados obtidos ao nível da aplicação através do *front-end* e da base de dados. Numa segunda fase, pretende-se demonstrar as interações entre os componentes intermédios do sistema através da análise do fluxo de informação entre os extremos da aplicação. Nesta, é ainda apresentado o protótipo em *breadboard* da *gateway* e do *end device*.

6.1.1 Resultados do Sistema Desenvolvido

Para a monitorização da informação obtida pelo sensor e para efetuar o controlo do atuador presentes no *end device* foi desenvolvida uma página *Web*. Esta página possui dois separadores que permitem visualizar e controlar estes dois componentes. Os dados obtidos do sensor são visíveis num destes separadores, denominado de "Data Log", através de uma tabela que contem três parâmetros, o valor do sensor, a hora e a data do momento no qual esta informação chegou ao servidor. Nesta tabela são apenas colocados os últimos doze valores recebidos e na primeira linha é sempre colocado o valor mais recente. Num outro separador, designado de "Status - LED", e de forma similar, é possível observar o estado do atuador bem como a hora e a data no qual foi efetuado o pedido para a alteração do estado através de uma tabela. A organização desta tabela

é realizada de forma semelhante à dos dados do sensor. Neste separador são ainda incluídos dois botões que permitem ligar ou desligar o atuador. Na Figura 6.1 são apresentados ambos separadores da página *Web* desenvolvida. Do lado esquerdo encontra-se o separador referente aos dados do sensor e do lado direito o separador referente ao controlo do atuador.

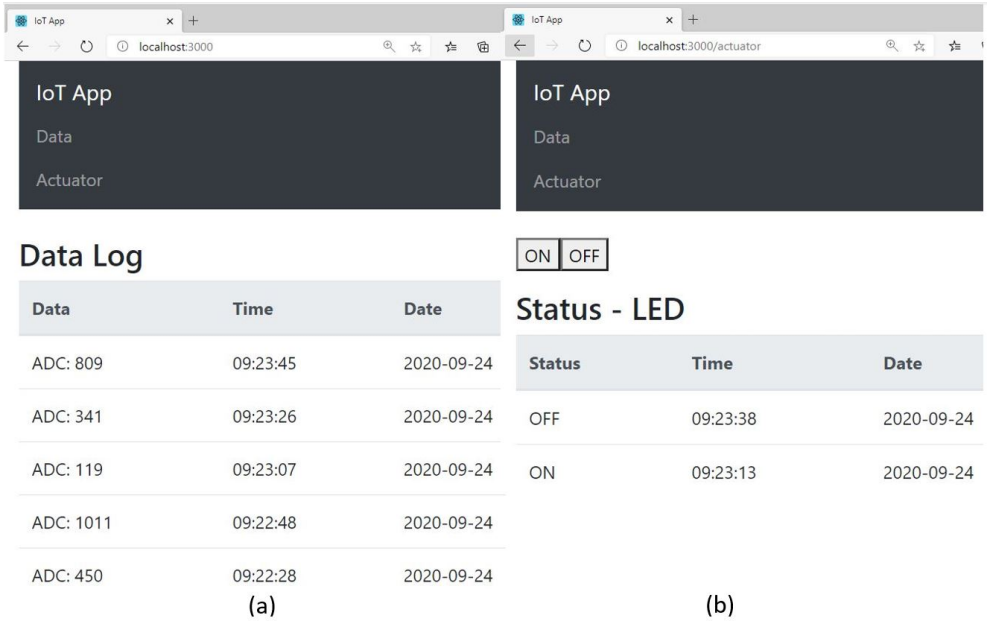


Figura 6.1: Separadores da página *Web* desenvolvida: (a) Separador Data Log e (b) Separador Status - LED.

Toda a informação disponibilizada pela página *Web* provém da base de dados, MongoDB. Como já foi referido, toda a informação encontra-se armazenada em duas coleções, uma para os dados obtidos do sensor e outra para o estado do atuador. O exemplo apresentado na Figura 6.2 pretende demonstrar como é armazenada a informação dentro de uma coleção na base de dados. Este exemplo é extraído da *shell* do MongoDB através do código apresentado na primeira linha, **db.datas.find()**. "**datas**" encontra-se assinalado na figura uma vez que se trata do nome da coleção que se pretende visualizar. Após a execução desta linha de código são disponibilizados todos os documentos referentes a esta coleção. Cada documento é composto por cinco campos no entanto apenas os primeiros três são relevantes para esta aplicação. O primeiro campo, denominado de "_id", representa um identificador único para este documento e é gerado automaticamente pela própria base de dados. Este campo é utilizado pelo *front-end* para organizar as tabelas. O segundo campo, "data", é onde é armazenado o valor proveniente do sensor. Por último, o campo "createdAt" serve para armazenar o dia e a hora. Estes últimos dois campos são posteriormente utiliza-

dos pelo *front-end* para a exibição dos dados. A coleção referente ao estado do atuador possui uma constituição idêntica à apresentada na Figura 6.2.

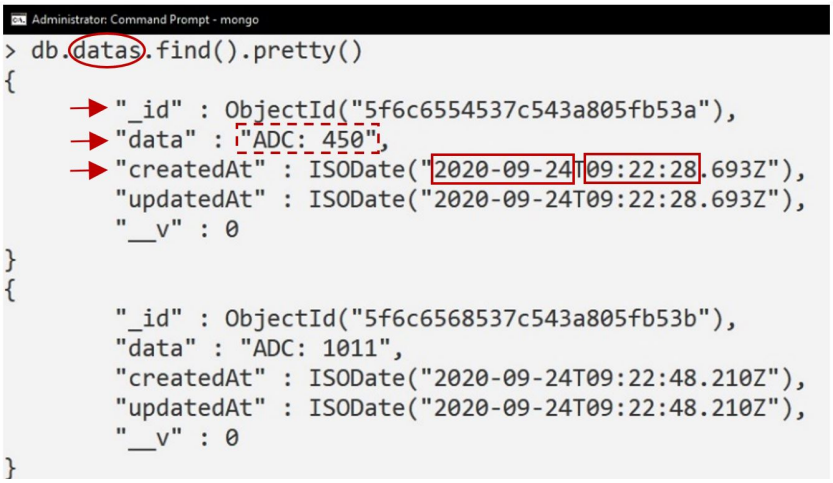


Figura 6.2: Armazenamento dos dados na base de dados MongoDB.

6.1.2 Fluxo de Informação no Sistema

Desde o *end device* até à base de dados a informação necessita de passar por dois pontos intermédios, nomeadamente a *gateway* e o servidor. A troca de informação entre estes dois componentes é realizada através do protocolo MQTT. Neste capítulo pretende-se visualizar as diferentes transições de informação nestes dois pontos. De modo a acompanhar visualmente estas transições foram impressas para a consola dos respetivos componentes mensagens que demonstram a ocorrência de determinados eventos. O fluxo de informação neste sistema apenas circula em dois sentidos, do *end device* para a base de dados e vice-versa. De modo a facilitar a compreensão estes serão analisados individualmente. A Figura 6.3 pretende demonstrar como atuam estes dois componentes caso o fluxo de dados circule no sentido do *end device* para a base de dados. As mensagens do lado esquerdo são produzidas pela *gateway* e as do lado direito referem-se ao servidor. Ainda nesta imagem, e antes de iniciar as trocas de dados, é possível verificar a correta inicialização e configuração de ambos os componentes através as mensagens presentes nos rectângulos a tracejado. A primeira interação entre os dois componentes ocorre ainda na inicialização da *gateway*, nomeadamente no estabelecimento da ligação MQTT. Estabelecida a ligação com sucesso, o *broker* gera uma mensagem que indica o nome do novo cliente.

A segunda interação, destacada na Figura 6.3 pelos retângulos preenchidos, diz respeito ao reencaminhamento de mensagens provenientes do *end device*. De modo a facilitar a compreensão das diferentes etapas será acompanhada a

mensagem "**ADC: 450**" gerada pelo *end device* após a leitura do sensor. No primeiro exemplo é possível observar que, na *gateway*, após a receção da mensagem LoRa com o conteúdo "**ADC: 450**", esta é posteriormente publicada no tópico "**datafromESP8266**" e, consequentemente, é enviada para o *broker*. Uma vez que durante a inicialização o pseudo-cliente presente no servidor subscreveu o tópico "**datafromESP8266**", a partir do momento em que o *broker* recebe uma mensagem neste tópico, este reencaminha-a imediatamente para o pseudo-cliente. O pseudo-cliente é depois responsável pelo armazenamento da mensagem na base de dados.

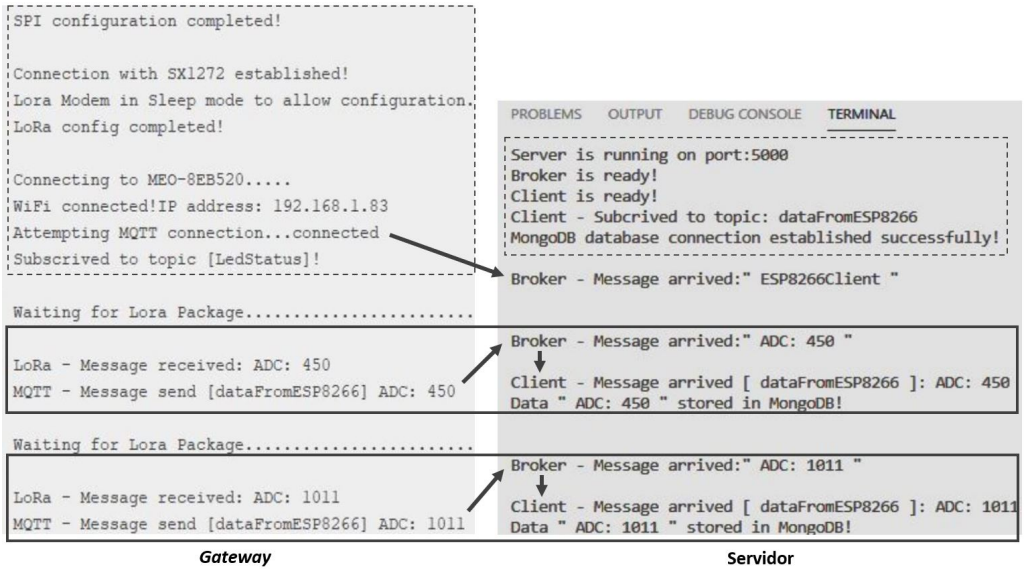


Figura 6.3: Demonstração do fluxo de dados na *gateway* e no servidor. Direção *end device* para base de dados.

A Figura 6.4 pretende demonstrar o percurso inverso, isto é, da base de dados para o *end device*. A alteração do estado do atuador inicia sempre com um pedido proveniente do *front-end*. Deste pedido resulta uma mensagem que é imediatamente armazenada na base de dados e posteriormente publicada pelo pseudo-cliente no tópico "**LedStatus**". Neste caso é demonstrado o percurso da mensagem "**ON**" originada no *front-end* e que se encontra destacada na Figura 6.4. Por sua vez a mensagem publicada é reencaminhada pelo *broker* para o cliente "ESP8266Client" presente na *gateway*, uma vez que este se encontra subscrito no tópico "**LedStatus**". A mensagem "MQTT - Message arrived [LedStatus] ON", gerada na *gateway*, indica que a mensagem foi recebida corretamente. Porém, esta não é imediatamente reencaminhada via LoRa para o *end device* uma vez que a janela de receção apenas abre após o envio de uma mensagem. Na Figura 6.4 encontra-se assinalada a mensagem "LoRa - Message send: ON" que

indica momento do envio da mensagem para o *end device*. É também possível observar que esta transmissão apenas ocorre após a receção de uma mensagem proveniente deste.

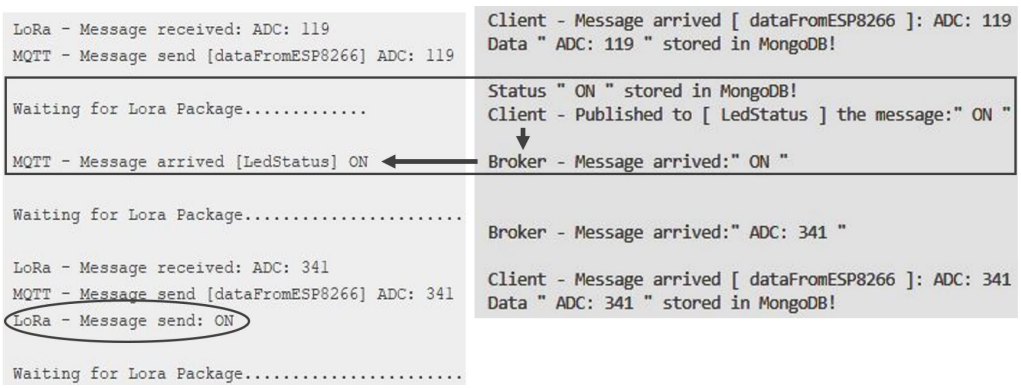


Figura 6.4: Demonstração do fluxo de dados na *gateway* e no servidor. Direção base de dados para o *end device*.

As Figuras 6.5 e 6.6 apresentam, respetivamente, o protótipo em *breadboard* da *gateway* e do *end device* desenvolvidos. Nestas é também possível observar os seus diferentes constituintes.

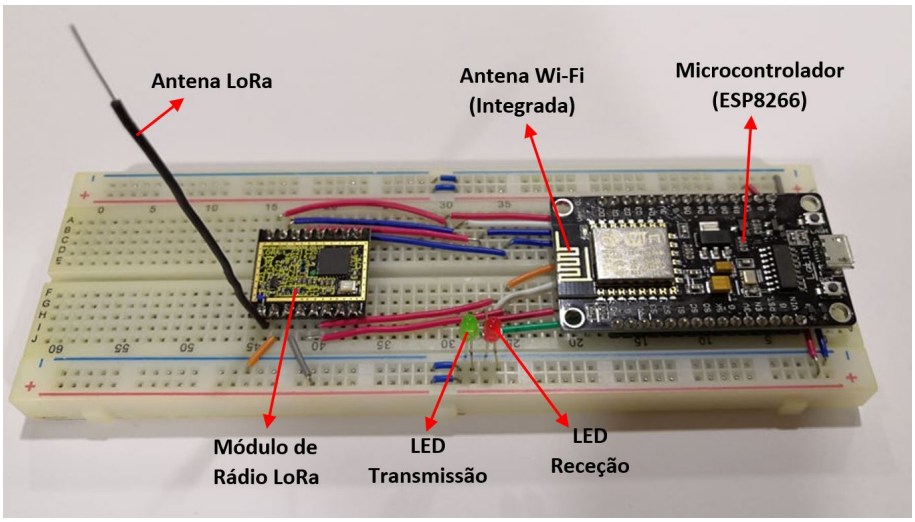


Figura 6.5: Protótipo em *breadboard* da *gateway*.

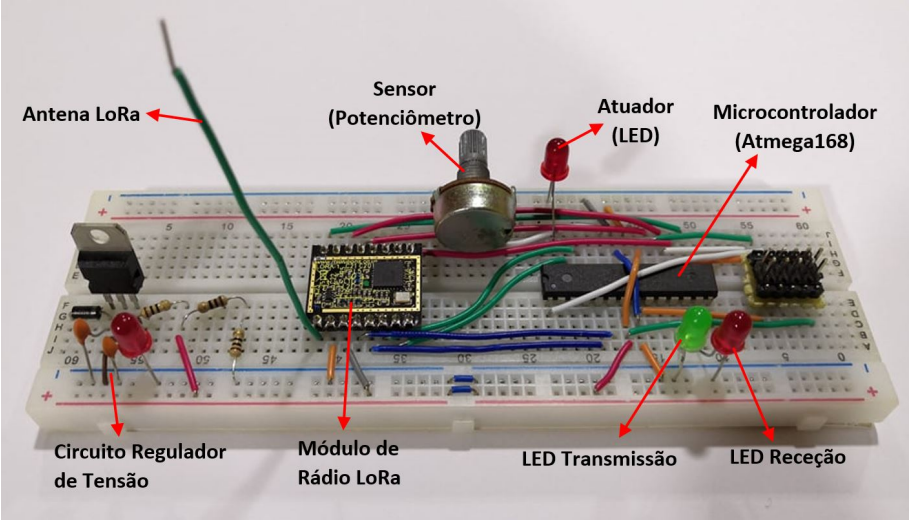


Figura 6.6: Protótipo em *breadboard* do *end device*.

6.2 Alcance Físico do Sistema

Uma vez verificado o correto funcionamento do sistema procedeu-se à avaliação do alcance físico do mesmo. Este teste tem como objetivo principal avaliar o alcance e consequentemente a fiabilidade do sistema de comunicação LoRa.

A análise do alcance do sistema de comunicação LoRa consistiu na colocação do *end device* a diferentes distâncias da *gateway* e efetuar a transferências de mensagens entre estes, avaliando posteriormente a sua correta transmissão e receção. Para isto foram realizados dois ensaios para cada distância definida. Em cada um dos ensaios, e tendo o *end device* como ponto de referência, foram enviadas cinco mensagens e foi aguardado pela receção de outras cinco. Após isto foi calculada a taxa de sucesso para cada um dos ensaios, tanto para o envio como para a receção. De seguida foi ainda efetuada a média entre o ensaios de modo a facilitara análise dos dados. Na Figura 6.7 encontra-se representado num mapa a localização dos diferentes pontos onde este teste foi realizado assim como a respetiva distância à *gateway*.

É importante referir que foram utilizados parâmetros de configuração do módulo de rádio LoRa presentes na Tabela C.1, referidos na secção 5.1.2. Dentro destes destaca-se o *spreading factor* de 11 e a *bandwith* de 125 kHz, uma vez que este parâmetros possuem influência direta no alcance do sistema. Além disso, esta experiência foi efetuada em ambiente urbano com infraestruturas de média e baixa dimensão. Os resultados obtidos desta experiência encontram-se resumidos na Tabela 6.1.

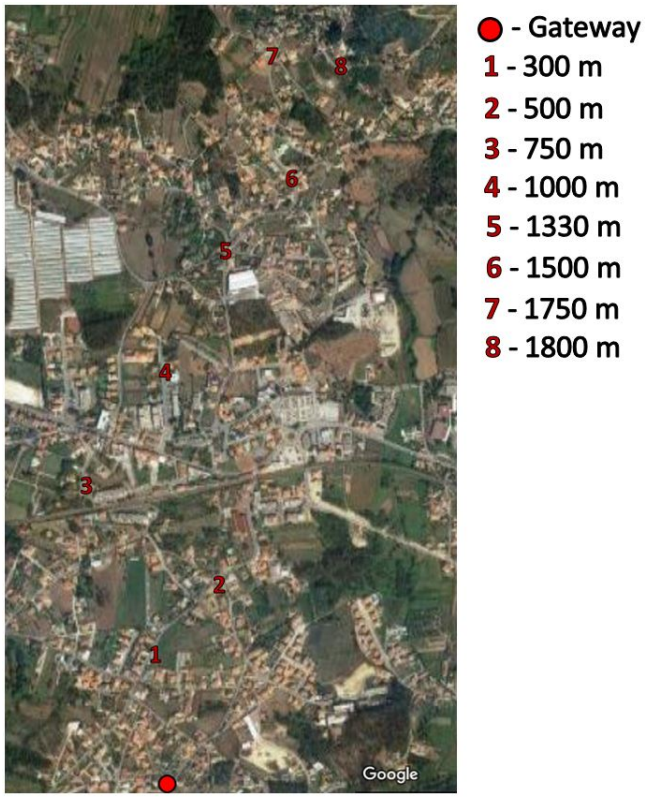


Figura 6.7: Mapa com a localização do pontos onde foram realizados os ensaios.

Tabela 6.1: Resultados do teste referente ao alcance físico do sistema.

Distância (m)	**Direção (Ponto de referência: ED)**	**Taxa de sucesso (%)**		
		Ensaio 1	**Ensaio 2**	**Média**
300 m	Envio	100	100	100
	Receção	100	100	100
500 m	Envio	100	100	100
	Receção	100	100	100
750 m	Envio	100	100	100
	Receção	100	100	100
1000 m	Envio	100	100	100
	Receção	100	100	100
1330 m	Envio	100	80	90
	Receção	60	100	80
1500 m	Envio	40	40	40
	Receção	40	0	20
1750 m	Envio	20	40	30
	Receção	0	40	20
1800 m (Direto)	*Envio*	*100*	*100*	*100*
	Receção	*100*	*100*	*100*

Através da análise da Tabela 6.1 é possível observar que até a uma distância de 1 km, a taxa de sucesso no envio e recção de pacotes de dados foi de 100 %. Porém, no ponto seguinte, localizado aproximadamente a 1,3 km da *gateway*, a taxa de sucesso reduziu ligeiramente em cerca de 10 % a 20 %, indicando a perda de alguns pacotes. No pontos seguintes, nomeadamente a 1,5 km e 1,75 km, nota-se uma queda abrupta na taxa de sucesso onde os valores são inferiores a 40 %. Os últimos dois ensaios, realizados a 1,8 km de distância apresentam uma taxa de sucesso de 100 %. Isto deve-se ao facto de serem realizados em condições diferentes dos restantes. Neste caso o posicionamento do *end device* encontrava-se na linha de visão direta da *gateway*, provando que nestas condições e até 1.8 km o sistema é capaz de funcionar sem perda de pacotes. No entanto este último valor é pouco relevante uma vez que o sistema desenvolvido será implementado numa localização com uma certa densidade de infraestruturas em seu redor tornando os restantes valores mais significativos para a avaliação do sistema numa aplicação em ambiente real.

Com os resultados obtidos neste teste e nas condições em que foi efetuado é possível concluir que até a uma distância de 1 km entre os transceptores a comunicação entre estes apresenta um nível de fiabilidade muito alto. Uma vez que o sistema de comunicação LoRa não dispõe de mecanismos de controlo de erros conclui-se que até uma distância de 1 km entre os transceptores o sistema apresentará um nível de fiabilidade alto.

Entre este ponto e o seguinte, localizado a 1,33 km obteve-se um nível de fiabilidade alto no entanto inferior ao anterior. A utilização da aplicação a esta distância iria resultar na perda de alguns pacotes de dados. Porém, com a implementação de algum tipo de mecanismo de controlo de erros a utilização deste sistema de comunicação poderá ser viável.

Nos pontos seguintes, localizados a 1,5 km e 1,75 km apresentam um nível baixo de fiabilidade na comunicação. A utilização do sistema desenvolvido a esta distância seria inviável e, mesmo com a implementação de mecanismos de controlo de erros, o número de retransmissões poderá ser elevado prejudicando significativamente a eficiência energética do *end device*, sendo este um dos focos principais no desenvolvimento deste.

Estas conclusões foram retiradas segundo os resultados obtidos nas condições já referidas, no entanto é importante referir que existem outros fatores que poderão ter influência no desempenho do sistema de comunicação LoRa, como por exemplo, a localização da *gateway*, a utilização de antenas de melhor qualidade em ambos os transceptores e a utilização de diferentes parâmetros de modulação LoRa. Cada um destes fatores poderá ter alterações mais ou menos significativas no desempenho deste sitema, no entanto neste trabalho as suas influências não foram avaliadas.

6.3 Análise do *Duty Cycle* e Tempo de Resposta do Sistema de Comunicação LoRa

O *duty cycle* e o tempo de resposta são dois parâmetros importantes no sistema desenvolvido. O primeiro indica a percentagem de tempo no qual o *end device* se encontra em trabalho, isto é, se encontra na fase de maior consumo energético. Caso se pretenda boa eficiência energética, o valor do *duty cylce* deverá ser o mais baixo possível. O cálculo do *duty cycle* resulta da divisão entre tempo no qual o dispositivo encontra-se a efetuar trabalho pelo tempo total de um ciclo. Neste caso, em relação ao tempo em que o sistema se encontra efetivamente em trabalho, apenas será considerado o tempo de transmissão de uma mensagem e o tempo de *timeout* para a receção de uma mensagem. No que diz respeito ao tempo de um ciclo, são contabilizados estes dois tempos já referidos bem como o período de tempo no qual o *end device* se encontra em modo *sleep*. Deste modo obteve-se a expressão 6.1 para o cálculo do *duty cycle*:

$$DutyCycle = \frac{T_{Transmission} + T_{Timeout}}{T_{Transmission} + T_{Timeout} + T_{Sleep}} \tag{6.1}$$

Em relação a cada um destes componentes temporais é importante referir que apenas o T_{Sleep}, período no qual o *end device* se encontra em modo *sleep*, é atribuído diretamente. No que diz respeito ao cálculo do período de *timeout* este já foi abordado na secção 3.5.3.5. Porém a expressão utilizada é a seguinte:

$$T_{Timeout} = SymbTimeout \cdot T_{Symbol} = SymbTimeout \cdot \frac{2^{SF}}{BW} \tag{6.2}$$

Relativamente ao tempo de transmissão ($T_{Transmission}$), segundo o *datasheet* [92], resulta da multiplicação do período por *symbol* com o somatório dos *symbols* presentes no *preamble* e na *payload*. É importante referir que nesta expressão *symbols* do *header* encontram-se incluídos na *payload*. A expressão 6.3 diz respeito ao cálculo do tempo de transmissão.

$$T_{Transmission} = (n_{Preamble} + 4.25 + n_{Payload}) \cdot T_{Symbol} \tag{6.3}$$

Por sua vez, o número de *symbols* presentes na *payload* é obtido a partir da equação 6.4.

$$n_{Payload} = 8 + max\left(ceil\left[\frac{8PL - 4SF + 28 + 16 - 20IH}{4(SF - 2DE)}\right](CR + 4), 0\right) \tag{6.4}$$

Nesta equação, PL corresponde ao número de *bytes* da *payload* (1 a 255) e o SF ao *spreading factor* (7 a 12). IH indica presença de um *header* implícito (IH = 0, *header* explícito, IH = 1, *header* implícito), CR refere-se ao *coding rate* (1 a 4,

onde 1 = 4/5 e 4 = 4/8) e o DE que indica a utilização do *Low Data Rate Optimize* (DE = 1, indica a sua utilização, 0 não). O *Low Data Rate Optimize* adiciona um nova camada de redundância na *payload* de modo a tornar a comunicação mais robusta. Este deve ser utilizado para comunicações em que o T_{Symbol} é superior a 16 ms. Isto acontece quando se utiliza uma BW de 125 kHz e um SF de 11 ou 12.

No que diz respeito ao tempo de resposta, neste caso, corresponde ao período de tempo entre a receção de duas mensagens. Por outras palavras, o tempo de resposta do sistema corresponde ao tempo de um ciclo que resulta do somatório dos tempos de transmissão, *timeout* e *sleep*. Quanto maior este período, maior será o tempo de resposta do sistema.

$$T_{Response} = T_{Transmission} + T_{Timeout} + T_{Sleep} \tag{6.5}$$

Através de uma breve análise das fórmulas apresentadas é possível observar que existe uma proporcionalidade inversa entre o *duty cycle* e o tempo de resposta do sistema. Isto leva a que um aumento do tempo de resposta resulta numa diminuição do *duty cycle* e vice versa. A partir destas fórmulas também foi possível concluir que de modo a obter-ser um *duty cycle* baixo assim como um tempo de resposta relativamente baixo é necessário que os tempos de transmissão e de *timeout* sejam os mais baixos possíveis.

Com os parâmetros de configurações do módulo de rádio LoRa utilizados no teste do alcance físico, e também já referidos na secção 5.1.5, resulta um *duty cycle* de aproximadamente 19 % e de um tempo de resposta de cerca de 18.6 segundos, onde o período de *timeout* é de 3 segundos e o tempo de transmissão é de 594 ms. Partindo deste valores, procurou-se diminuir o tempo de resposta tentando simultaneamente manter um valor do *duty cycle* baixo.

Numa primeira fase diminui-se o tempo de *timeout* para o menor valor possível. Através da expressão 6.2 e com um SF de 11 e com uma BW de 125 kHz, o menor período de *timeout* permitido pelo módulo de rádio LoRa, é de 65.6 ms. Este é obtido utilizando o menor valor possível para *SymbTimeout*, 4. Porém, com este período de *timeout* o sistema desenvolvido não é capaz de captar as mensagens provenientes da *gateway*. No entanto com a utilização de um *SymbTimeout* de 5, que resulta numa período de *timeout* de 82 ms, o sistema desenvolvido já é capaz de captar as mensagens procedentes da *gateway*. Desta forma, para o sistema desenvolvido o período de *timeout* mínimo é de 82 ms para um SF de 11 e uma BW de 125 kHz. A utilização de períodos de *timeout* tão pequenos levantou um problema a nível prático mais propriamente no princípio de funcionamentos de ambos os transceptores. Devido a estes tempos diminutos é necessário tornar as transições entre os modo de transmissão e receção no *end*

device e vice-versa na *gateway* o mais rápido possível e com um certo nível de sincronismo, de forma a garantir que a mensagem envidada pela *gateway* possa ser captada pelo *end device* durante esta janela de receção.

Definido o período de *timeout*, e numa segunda fase, foi obtido o tempo de transmissão a partir das configurações utilizadas. É importante referir que para a simulação da *payload* foi utilizada a *string* "ADC: 1023" que corresponde a 9 *bytes*. Com estes parâmetros obteve-se um tempo de transmissão de 593.9 ms. De forma a poder observar a influência destas alterações foi mantido o *duty cycle* de 19 %, que levou a uma alteração do T_{Sleep}, e foi comparado o tempo de resposta do sistema. Com estes períodos de transmissão e *timeout* foi possível reduzir o T_{Sleep} de 15 segundos para 2.9 segundos e o tempo de resposta do sistema passou de aproximadamente 18 segundos para 3.6 segundos.

Apesar de o tempo de resposta do sistema ter sido reduzido significativamente com a diminuição do período de *timeout*, este poderá ser insuficiente para determinadas aplicações. No entanto com a manipulação de alguns parâmetros relacionados com o pacote de dados LoRa e com sua modelação, e que não influenciam o alcance do sistema, é possível reduzir o tempo de transmissão. Nesta situação, e de modo a poder verificar a sua influência no *duty cycle*, foi definido um tempo de resposta do sistema fixo de 1.5 segundos. Nos ensaios de 1 a 7 da Tabela 6.2 pretendem resumir a influência que cada uma destas alterações provoca no período de transmissão e no *duty cycle*. Além disso, procurou-se descobrir qual seriam as configurações necessárias para se obter o menor tempo de transmissão possível sem que estas influenciem o alcance do sistema.

No ensaio 1 temos as configurações iniciais, já referidas, que resultam num período de transmissão de 593.9 ms e num *duty cycle* de 45.1 %, para um tempo de resposta de 1.5 segundos. De imediato é possível observar que estas configurações apresenta um *duty cycle* elevado. No ensaio seguinte, de modo a diminuir o período de transmissão foi reduzido o CR para menor valor disponível. Este permitiu aumentar o CCR de $4/8$ para $4/5$ diminuindo significativamente o *overhead* da transmissão, comprometendo porém a robustez da mesma. Esta alteração permitiu diminuir o período de transmissão para 495.6 ms e o *duty cycle* para 38.5 %.

No que diz respeito aos parâmetros de modulação LoRa, o CR é o único que permite ser modificado sem que o alcance da comunicação seja impactado. Deste modo, de forma a diminuir o tempo de transmissão é necessário reduzir o tamanho do pacote LoRa. Numa primeira fase, no ensaio 3, foi colocado menor tamanho possível para o *preamble*. Este passou de 12.25 *bytes* para 10.25 *bytes*, que levou a uma ligeira diminuição do tempo de transmissão para 462.9 ms. No ensaio 4, procedeu-se à diminuição do tamanho da *payload* para 4 *bytes*, simulando apenas o envio da informação essencial, neste caso a *string* "1023". Neste

ensaio foi possível diminuir o período de transmissão para 380.9 ms e o *duty cycle* para 30.9 %. No ensaio 5 e 6, os resultados foram idênticos ao ensaio 5, no entanto as abordagens foram diferentes. No ensaio 5 foi utilizado um *header* implícito e não se optou pena utilização de um CRC na *payload* o que permitiu aumentar 5 *bytes* ao tamanho da *payload* sem que período de transmissão se alterasse. No ensaio 6, foi utilizada uma *payload* de 4 *bytes* com CRC e com um *header* implícito, no entanto o período de transmissão manteve-se. Isto deve-se ao facto de que nestes três ensaios o número total de *symbols* da *payload* ser igual apesar de terem sido utilizadas configurações diferentes.

No ensaio 7 foi onde foi obtido o menor valor possível para o período de transmissão e *duty cycle*. No ensaio 7 foi excluído o *header* e o CRC da *payload* e foi utilizada a *payload* reduzida. Nesta configuração foram obtidos os melhores resultados obtendo-se um período de transmissão e *duty cycle* de, respetivamente, 299 ms e 25.4 %.

Comparativamente às configurações iniciais é possível observar uma redução importante de 19.7 % no *duty cycle*. Esta redução, apesar de ser relativamente pequena, irá contribuirá fortemente ao nível da eficiência energética do *end device*.

Tabela 6.2: Análise do *duty cycle* e tempo de resposta do sistema de comunicação LoRa.

Nº Ensaio	SF	BW (kHz)	CCR	T_{Symbol} (ms)	Dados do pacote LoRa						$T_{Transmission}$ (ms)	$T_{Timeout}$ (ms)	T_{Sleep} (ms)	*Duty Cycle (%)* ($T_{Response}$ = 1.5 s)
					$n_{Preamble}$ (Symbols)	Header	Payload (bytes)	CRC	DE	$n_{Payload}$ (Symbols)				
1	11	125	4/8	16.4	12.25	✓	9	✓	✓	24	593.9	82	824.1	45.1
2	11	125	**4/5**	16.4	12.25	✓	9	✓	✓	18	495.6	82	922.4	38.5
3	11	125	4/5	16.4	**10.25**	✓	9	✓	✓	18	462.9	82	955.1	**36.3**
4	11	125	4/5	16.4	10.25	✓	**4**	✓	✓	**13**	380.9	82	1037.1	30.9
5	11	125	4/5	16.4	10.25	×	**9**	×	✓	**13**	380.9	82	1037.1	30.9
6	11	125	4/5	16.4	10.25	×	**4**	✓	✓	**13**	380.9	82	1037.1	30.9
7	11	125	4/5	16.4	10.25	×	**4**	×	✓	8	299	82	1119	**25.4**
Valor máximo e mínimo com um SF de 10														
8	**10**	125	4/5	8.2	10.25	✓	9	✓	×	18	231.4	41	1227.6	18.2
9	**10**	125	4/5	8.2	10.25	×	4	×	×	8	149.5	41	1309.5	12.7
Valor máximo e mínimo com uma BW de 250 kHz														
10	11	**250**	4/5	8.2	10.25	✓	9	✓	×	18	231.4	41	1227.6	18.2
11	11	**250**	4/5	8.2	10.25	×	4	×	×	8	149.5	41	1309.5	12.7
Valor máximo e mínimo com um SF de 10 e uma BW de 250 kHz														
12	**10**	**250**	4/5	4.1	10.25	✓	9	✓	×	18	115.7	20.5	1363.8	9.1
13	**10**	**250**	4/5	4.1	10.25	×	4	×	×	8	74.8	20.5	1404.7	6.4

Adicionalmente foram realizados outros seis ensaios, no entanto, ao contrário do que aconteceu nos anteriores, os parâmetros manipulados foram o SF e a BW. Em relação aos restantes parâmetros, nomeadamente CCR e composição do pacote LoRa, foram utilizados os dos ensaios 3 e 7 de modo a avaliar os extremos ao nível do *duty cycle* e do tempo de transmissão, contudo será atribuída maior ênfase aos valores mínimos. A desvantagem da manipulação do SF e da BW é a alteração do alcance do sistema. Por outro lado, a alteração destes parâmetros permite modificar o período por *symbol*, T_{Symbol}, e consequentemente a taxa de transmissão. Esta pode ser aumentada através da diminuição do SF, do aumento da BW, ou ambos simultaneamente. Cada uma destas situação será analisada individualmente no entanto apenas será reduzido ou aumentado um patamar em cada um destes parâmetros de modo a ter o menor impacto possível no alcance. É importante referir que a diminuição do alcance resultante da alteração destes parâmetros não foi analisada.

Nos ensaios 8 e 9, apenas o SF foi diminuído para 10 o que permitiu diminuir para metade o período por *symbol* de 16.4 ms para 8.2 ms. Consequentemente isto permitiu reduzir para metade o tempo de transmissão e o *duty cycle* em ambas as situações. O valor mínimo, com um SF de 10, para o tempo de transmissão e para o *duty cycle* foi de, respetivamente 231.4 ms e 18.2 %. É importante referir que, contrário do que aconteceu nos ensaios anteriores, de 1 a 7, a alteração do SF ou BW resulta também numa modificação do tempo de *timeout*. Com um SF de 10 e uma BW de 125 kHz, este tempo é reduzido para 41 ms.

No que diz respeito aos ensaios 10 e 11, apenas o BW foi aumentado para 250 kHz. Apesar de se ter alterado um parâmetro diferente os resultados obtidos foram idênticos aos ensaios 8 e 9. Deste modo, o aumento da BW para o dobro, permitiu reduzir o período por *symbol* para 41 ms resultando numa taxa de transmissão e *duty cycle* de, respetivamente, 149.5 ms e 12.7 % O período de *timeout* manteve-se igual aos ensaios 8 e 9 com um valor de 41 ms.

Nos ensaios 12 e 13 o SF foi reduzido para 10 e BW aumentada para 250 kHz. A utilização destes parâmetros permitiu reduzir o período por *symbol* para 4.1 ms, cerca de metade do tempo obtido nos ensaios anteriores. De igual forma ao que aconteceu no ensaios anteriores, tanto o período de transmissão e o *duty cycle* evoluíram para metade, 74.8 ms e 6.4 %. No que diz respeito ao período de *timeout* este foi igualmente reduzido para metade, 20.5 ms.

Com a realização desta análise é possível verificar a maior ou menor influência que cada um dos parâmetros de configuração do módulo de rádio LoRa provoca ao nível do *duty cycle* e tempo de resposta do sistema. A partir do ensaio 2 é possível observar o *overhead* introduzido pelo CR e o seu impacto ao nível do tempo de transmissão. Caso a aplicação no qual será inserido este sistema necessite de um tempo de reposta assim como um duty cycle baixo é aconselhável

a utilização do CR mais baixo possível desde que não comprometa a robustez da comunicação. Com os ensaios de 3 a 7 revelou-se a importância de um pacote LoRa otimizado, uma vez que a sua dimensão tem impacto direto no tempo de transmissão e consequentemente no *duty cycle* e no tempo de resposta do sistema. Recomenda-se a utilização de um *header* implícito sempre que possível, de modo a evitar o envio de informação redundante em cada pacote. Além disso o formato dos dados na *payload* deve ser o mais eficiente possível de modo a diminuir o seu tamanho. Em aplicações muito sensíveis tem termos de tempo de resposta e *duty cycle* a remoção do CRC na *payload* poderá ser uma opção.

Através da análise do ensaios de 8 a 13 é possível observar que, a alteração do SF ou da BW provoca, um aumento para dobro ou uma redução para metade do período por *symbol* e consequentemente do tempo de transmissão de um pacote de dados. Da mesma forma o período de *timeout* é alterado. Apesar de esta análise apenas se ter realizado para um SF de 10 e 11 e para uma BW de 125 kHz e 250 kHz, é previsível que esta tendência mantêm ao longo dos diferentes valores disponíveis para o SF e para a BW. Em aplicações que onde é possível a redução do alcance em prol do tempo de transmissão, estas alterações devem ser efectuadas uma vez que irá benificiar a eficiência energética do *end device*.

Concluindo, após a realização de todos estes ensaios é possível verificar a importância da correta configuração de todos os parâmetros relativos à modulação LoRa e à respetiva constituição do pacote de dados. Porém, serão as necessidades impostas pela aplicação que irão ditar quais os requisitos necessários ao nível do *duty cycle* e tempo de resposta do sistema, e consequentemente quais as melhores configurações a serem aplicadas para esse fim. No entanto, a tecnologia LoRa permite a modificação de diversos parâmetros que a torna versátil e com a capacidade de ser aplicada numa grande variedade de aplicações.

6.4 Regulamento de Utilização das Bandas de Frequência ISM e as suas Implicações no Tempo de Resposta

Apesar de as bandas de frequência ISM serem isentas de taxas de utilização, estas necessitam de seguir determinadas regras estabelecidas pelos governos de cada país ou região. No caso da Europa um dos documentos regulatórios é o CEPT Rec. 70-03 [100] e a presente secção terá por base este documento. Estas restrições foram implementadas com a intenção de minimizar as interferências nestas bandas de frequência. O objetivo desta secção não é realizar uma análise detalhada deste documento, mas apenas alertar para a sua existência e para a influência que estas restrições podem ter no sistema [100][101][102].

Uma das restrições que este documento impõe para minimizar as interferências, e a única que será abordada nesta secção, é o estabelecimento de um limite máximo de *duty cycle* de ocupação do meio, $DutyCycle_{Transmission}$. É importante não confundir o *duty cycle* de ocupação do meio com o *duty cycle* abordado na secção 6.3, uma vez que este último diz respeito ao *duty cycle* de funcionamento do *end device* indicando a percentagem de tempo no qual se encontra na fase de maior consumo energético, enquanto que o *duty cycle* de ocupação do meio indica a percentagem de tempo no qual o dispositivo se encontra a transmitir, ou seja, a ocupar o meio. Deste modo, o cálculo deste *duty cycle* de ocupação do meio é dado pela expressão 6.6.

$$DutyCycle_{Transmission} = \frac{T_{Transmission}}{T_{Transmission} + T_{Timeout} + T_{Sleep}} \tag{6.6}$$

Este documento divide esta banda ISM em sub bandas menores com valores de *duty cycle* de ocupação do meio diferentes. Neste caso, como foi utilizada uma frequência portadora de 869.5 MHz, esta fica incluída no intervalo de frequências entre os 869.4-869.65 MHz e que possibilita um *duty cycle* de ocupação do meio de até 10 %. Porém, outras sub bandas como por exemplo entre os 868-868.6 MHz ou entre os 868.7-869.2 MHz permitem apenas *duty cycles* de 1 % e 0.1 %, respetivamente.

Esta secção tem como objetivo avaliar as implicações que esta restrição do *duty cycle* provoca no tempo de resposta do sistema. No entanto apenas será avaliado para a frequência portadora utilizada de 869.5 MHz. A Tabela 6.3 pretende demonstrar o tempo de resposta do sistema para um *duty cycle* de ocupação do meio de 10 % para diferentes configurações de modelação e do pacote LoRa. Nesta Tabela são apenas avaliados os valores máximos e mínimos, com as características do pacote LoRa apresentadas na Tabela 6.2. De modo a facilitar a visualização destas caraterísticas, o números dos ensaios da Tabela 6.2 correspondem aos números dos ensaios da Tabela 6.3.

A partir da Tabela 6.3 é possível observar que, nos ensaios 3 e 7, para um SF de 11 e uma BW de 125 kHz, o sistema apresenta um tempo de resposta máximo e mínimo de aproximadamente 4.6 segundos e 3 segundos, respetivamente. Para determinadas aplicações este tempo de resposta pode ser suficiente, porém para outras este período pode ser demasiado longo. Deste modo, nos ensaios 8 e 9, o SF foi reduzido para 10 provocando uma diminuição período de resposta para cerca de metade comparativamente aos ensaios 3 e 7. Em contrapartida o alcance do sistema diminui. Assim, nos ensaios 8 e 9, o valor máximo é de 2.3 segundos e valor mínimo é de cerca de 1.5 segundos. Por último, nos ensaios 12 e 13, foi avaliado o tempo de resposta para um SF de 10 e uma BW de 250 kHz, obtendo-se períodos máximos e mínimos de aproximadamente 1.1 segundos e 747 ms, respetivamente. Mais uma vez, com o aumento da BW para

250 kHz foi possível diminuir estes valores para cerca de metade comparativamente aos obtidos nos ensaios 8 e 9. Com estas configurações os tempos de resposta do sistema foram significativamente mais baixos quando comparados com os ensaios 3 e 7. Com a alteração individual do SF ou da BW é possível observar um aumento para o dobro ou diminuição para cerca de metade dos tempos de resposta do sistema. Adicionalmente é possível verificar que, para todos os ensaios com as mesmas configurações de modulação, existe uma redução de cerca de 34 % do valor máximo para mínimo do tempo de resposta conseguida apenas pela alteração da configuração do pacote LoRa.

Tabela 6.3: Análise tempo de resposta do sistema de comunicação LoRa para um *duty cycle* de ocupação do meio de 10 % imposto pelo CEPT Rec. 70-0.

Nº Ensaio	SF	BW (kHz)	CCR	$T_{Transmission}$ (ms)	$T_{Timeout}$ (ms)	T_{Sleep} (ms)	$T_{Response}$ (*ms*) ($DutyCycle_{Transmission}$= *10%*)
Valor máximo e mínimo com um SF de 11 e uma BW de 125 kHz							
3	11	125	4/5	462.9	82	4048.1	4599.3
7	11	125	4/5	299	82	2609	2990
Valor máximo e mínimo com um SF de 10 e uma BW de 125 kHz							
8	10	125	4/5	231.4	41	2041.6	2314
9	10	125	4/5	149.5	41	1304.5	1495
Valor máximo e mínimo com um SF de 10 e uma BW de 250 kHz							
12	10	250	4/5	115.7	20.5	1020.8	1157
13	10	250	4/5	74.8	20.5	652.7	747.7

Esta análise permite concluir que esta limitação de *duty cycle* de ocupação do meio provoca, consequentemente, limitações no tempo de resposta do sistema. Porém, caso a aplicação em questão possua alguma liberdade de escolha em termos de alcance, com a manipulação de alguns dos parâmetros de modulação e com a configuração do pacote LoRa é possível diminuir o tempo de resposta significativamente. É importante salientar mais uma vez que a correta configuração do pacote LoRa permite reduções no período de resposta do sistema e que estas poderão ser fulcrais para que o sistema cumpra ou não o tempo de resposta imposto pela aplicação.

Capítulo 7

Conclusões

Neste capítulo são apresentadas as considerações finais relativas ao trabalho realizado. Além disso, serão abordadas as principais dificuldades sentidas e serão propostas algumas ideias para trabalhos futuros.

7.1 Considerações Finais

O objetivo principal desta dissertação consistiu no estudo da tecnologia LoRa e efetuar a sua integração num sistema *healthcare* simples baseado na filosofia da IoT. Uma vez que a tecnologia LoRa seria inserida neste conceito foi necessário, numa primeira fase, efectuar um estudo acerca do conceito da IoT e de algumas tecnologias associadas a este, bem como a aquisição de alguns conhecimentos relativos ao segmento da IoT dos cuidados médicos e de saúde. Além disso foi efetuado um estudo acerca da tecnologia LoRa que permitiu adquirir conhecimentos acerca do seu princípio de funcionamento e que posteriormente serviram de base para o sistema desenvolvido neste trabalho.

Seguidamente foi definida uma arquitetura para o sistema. Nesta foram definidos não só todos os componentes necessários para o desenvolvimento de um sistema *healthcare* baseado na filosofia da IoT, mas também outras tecnologias e ferramentas fundamentais para a implementação deste sistema.

Uma vez elaborado todo o trabalho de pesquisa, procedeu-se à implementação do sistema. Devido à complexidade imposta pela quantidade de componentes e tecnologias associadas a estes procedeu-se à divisão do trabalho em três sistemas distintos, o sistema de comunicação LoRa, o sistema de comunicação com o *back-end* e o sistema de ligação do *back-end* com *front-end*. De modo a facilitar a implementação, cada um destes sistemas foi desenvolvido individualmente e posteriormente integrados na solução final.

Seguidamente, esta solução foi submetida a alguns testes apresentando resultados bastante satisfatórios. O sistema de comunicação LoRa foi o foco principal do trabalho e foi onde grande parte do tempo utilizado para a realização deste foi despendido. Após a submissão deste sistema a alguns testes, este apresentou bons resultados e permitiu tirar algumas conclusões importantes acerca do mesmo. Relativamente ao alcance do sistema de comunicação LoRa este apresentou bons resultados. A sua capacidade de ultrapassar obstáculos é um dos pontos fortes desta tecnologia e, uma vez que a probabilidade de um sistema *healthcare* vir ser desenvolvido num ambiente *indoor* é elevada, torna a conjugação destes uma opção viável em aplicações futuras. No que diz respeito à taxa de transmissão de dados, a tecnologia LoRa possui uma taxa baixa comparativamente a outras tecnologias existentes no mercado. Esta caraterística torna-a inadequada para ser empregue em aplicações críticas da IoT, no entanto capaz de ser utilizada em aplicações em massa. Através dos resultados obtidos, ao nível da taxa de transmissão e respetivo tempo de resposta do sistema, foi concluído que deverão ser tidos em conta alguns aspetos relativamente ao uso desta tecnologia. A tecnologia LoRa permite a modificação de diversos parâmetros que permitem alterar o seu alcance e taxa de transmissão, porém existe uma relação contraditória entre estas duas caraterísticas. Deste modo, o equilibrio entre a taxa de transmissão e o alcance é um fator chave de modo a ser possível tirar o melhor partido desta tecnologia. Além disso, em aplicações bastante sensíveis em termos de tempo resposta, a correta otimização do pacote LoRa pode ter grande impacto, não só ao nível de desempenho do sistema mas também ao nível da eficiência energética. Contudo, é a aplicação sobre a qual será inserido este sistema e os requisitos impostos por esta que ditarão a sua viabilidade de ser utilizada para esse fim.

Relativamente aos sistemas de comunicação com o *back-end* e a sua ligação com o *front-end*, apesar de não terem sido o foco principal do trabalho, estes foram essenciais para o desenvolvimento deste sistema baseado na filosofia da IoT. No que diz respeito ao sistema de comunicação com o *back-end*, a utilização do protocolo MQTT apresentou-se como uma boa escolha, desempenhando de forma fiável a sua função. Ainda em relação ao *back-end*, a base de dados MongoDB apresenta uma simplicidade e flexibilidade na escalabilidade que a tornam apta de ser utilizada neste tipo de aplicações. Em relação ao sistema de ligação do *back-end* com o *front-end*, a utilização do protocolo HTTP satisfaz as necessidades impostas, porém não se apresenta como o protocolo mais adequado. Este não permite a actualização em tempo real dos dados presente na base de dados para o *front-end*, sendo para isso necessário realizar uma atualização periódica ao *front-end*. A escolha de um outro protocolo, como o *WebSockets*, poderia ter sido uma melhor opção. Porém, o tempo despendido nos outros componentes do trabalho não permitiu a sua implementação e avaliação

do mesmo.

Relativamente ao projeto como um todo, o sistema desenvolvido apresenta-se como um sistema com as característica necessárias para ser aplicado e adaptado para diversas soluções, não só no segmento dos cuidados médicos e de saúde da IoT, mas também em outros segmentos.

Concluindo, apesar da dificuldades encontradas durante o desenvolvimento desta dissertação, todos os objetivos estabelecidos inicialmente foram atingidos. Este trabalho permitiu a aquisição de conhecimentos na área da IoT e sobre diversas tecnologias que poderão vir a estar presentes no nosso quotidiano num futuro relativamente próximo. Além disso, espera-se que este trabalho possa vir a ser contribuição, maior ou menor, ou até mesmo uma ponto de partida para projectos semelhantes na mesma área ou até mesmo em outras áreas.

7.2 Principais Dificuldades Encontradas

A principal dificuldade sentida ao longo do trabalho revelou-se a nível prático com a integração dos diversos conceitos e tecnologias num só sistema. Este problema agravou-se especialmente no desenvolvimento do *back-end* e *front-end* devido à pouca experiência e conhecimentos relativos não só à linguagem de programação JavaScript, mas também com outras ferramentas como o NodeJS, ReactJS e as diferentes bibliotecas e *frameworks* associadas a estas. Por vezes, o avanço ou pequenas mudanças no trabalho, demoravam muito mais tempo do que seria esperado, tempo que poderia ter sido utilizado para melhorar outros aspetos do trabalho.

Relativamente à tecnologia LoRa a principal dificuldade prendeu-se com os diversos parâmetros de configuração associados a esta. Além disso, a existência de diversas dependências em relação a alguns dos parâmetros tornou, numa fase inicial, difícil a colocação dos módulos de rádio a funcionar de forma correta. Porém, após a sua compreensão esta dificuldade foi ultrapassada.

7.3 Ideias para Trabalhos Futuros

Como ideia para trabalho futuro, e já referida, seria substituição do protocolo HTTP presente no *front-end* por um outro protocolo, como por exemplo *WebSockets*, que permita uma atualização em tempo real dos valores presentes na base de dados para o *front-end*.

Uma outra ideia teria a ver com a escalabilidade do sistema. O aumento do mesmo através da introdução de mais *end devices* ou até mesmo de *gateways*. No entanto, a introdução de *end devices* implicaria uma modificação do modo

de funcionamento do sistema de comunicação LoRa, mais concretamente na *gateway*, de modo a poder receber pacotes provenientes de múltiplos *end devices*. Por outro, a introdução de novas *gateways* poderia ser mais fácil uma vez que o protocolo MQTT já se encontra preparado para lidar com múltiplos clientes.

Um outro ponto de evolução para este trabalho, e também já referido, seria a implementação de um mecanismo de controlo de erros no sistema de comunicação LoRa. A presença de um mecanismo deste género iria aumentar a fiabilidade do sistema e permitir a sua utilização em alcances superiores.

Outra ideia para trabalho futuro seria a avaliação do alcance do sistema para diferentes configurações, com especial incidência no *spreading factor* e a *bandwidth*, de modo a permitir ao utilizador saber, de forma mais concreta, qual o alcance que o sistema teria para determinadas taxas de transmissão.

Por último, poderia ser realizada uma otimização e avaliação do consumo energético efectivo do *end device*. Além disso poderia ser efetuada uma estimativa da longevidade do *end device* para uma determinada bateria.

Referências Bibliográficas

[1] "Internet of things (iot)." Obtido de US Communications and Eletric Inc. https://www.uscande.com/iot.php. Acedido: 11 de novembro de 2019. [Quoted on p. 1, 7]

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, pp. 2347–2376, 2015. [Quoted on p. 1, 29]

[3] M. K. B. H. H. Perwej, Y; Aboughaly, "An extended review on internet of things (iot) and its promising applications," *CAE*, fevereiro de 2019. [Quoted on p. 5, 7]

[4] T. I. of Electrical and E. Engineers, *IEEE Standards Association (IEEE-SA) Internet of Things (IoT) Ecosystem Study*. IEEE, Inc, 2015. [Quoted on p. 5]

[5] P. M. Keyur, P. K. ; Sunil, "Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application and future challenges," *IJESC*, 2016. [Quoted on p. 5]

[6] "Why the internet of things is called internet of things: Definition, history, disambiguation." Obtido de IoT Analytics https://iot-analytics.com/internet-of-things-definition/. Acedido: 14 de novembro de 2019. [Quoted on p. 6]

[7] S. E. Park, "Internet of thing: An introduction," *Congressional Research Service*, 4 de junho de 2019. [Quoted on p. 7]

[8] "The top 10 iot segments in 2018." Obtido de IoT Analytics https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/. Acedido: 13 de novembro de 2019. [Quoted on p. 8, 9]

[9] Cisco, "Cisco connected mining." Obtido de Cisco: https://www.cisco.com/c/dam/en_us/solutions/industries/automotive/docs/connected_mining_at-a-glance.pdf. Acedido: 25 de abril de 2020. [Quoted on p. 8]

[10] S. Eletric, "Sustainable hotels, marriot china." Obtido de Shneider Eletric: https://www.se.com/us/en/work/campaign/life-is-on/case-study/marriott-hotel-china.jsp. Acedido: 25 de abril de 2020. [Quoted on p. 9]

[11] S. Eletric, "Marina bay sands singapure." Obtido de Shneider Eletric: https://www.se.com/us/en/work/campaign/life-is-on/case-study/marina-bay-sands.jsp. Acedido: 25 de abril de 2020. [Quoted on p. 9]

[12] FleetNest, "Fleetnest." Obtido de FleetNest: https://www.fleetnest.com/. Acedido: 25 de abril de 2020. [Quoted on p. 9]

[13] Telti, "Irrigation management." Obtido de Telti: https://www.telit.com/industries-solutions/agriculture/irrigation-management/. Acedido: 26 de abril de 2020. [Quoted on p. 9]

[14] Telti, "Croplivestockmanagement." Obtido de Telti: https://www.telit.com/industries-solutions/agriculture/crop-livestock-monitoring/. Acedido: 26 de abril de 2020. [Quoted on p. 9]

[15] A. Business, "Iot package tracking solution." Obtido de AT Business: https://www.business.att.com/products/package-tracking.html. Acedido: 26 de abril de 2020. [Quoted on p. 9]

[16] K. C. Wiklundh, "Understanding de iot technology lora and its interference vulnerability," *EMC Europe*, setembro de 2019. [Quoted on p. 10]

[17] A. Labs, "Iot cellular network." Obtido de Altice Labs https://www.alticelabs.com/content/WP-IoT-Cellular-Networks.pdf. Acedido: 16 de janeiro de 2020. [Quoted on p. 10, 18, 23]

[18] Y. Duroc, "On the system modeling of antennas," *Progress In Electromagnetics Research (PIER) Journal*, vol. 21, 04 2010. [Quoted on p. 11]

[19] R. Silva, P. Machado, R. Rocha, and S. Silva, "The light and the sunscreens: A social-theme," *Revista Virtual de Química*, vol. 7, 01 2015. [Quoted on p. 11]

[20] D. Humphreys, D. Rhodes, D. Barker, D. Happy, D. Lund, F. Hamid, G. Varrall, G. Macdonald, J. Lota, M. Barrett, M. Beach, M. Fitch, M. Geen, M. Dohler, M. Juwad, P. Jenkins, Q. Ni, R. Tafazoli, R. Sivalingam, and Z. Ding, "5g innovation opportunities- a discussion paper," 08 2015. [Quoted on p. 12]

[21] "Wireless communication: Introduction, types and applications." Obtido de Electronics Hub: `https://www.electronicshub.org/wireless-communication-introduction-types-applications`. Acedido: 9 de março de 2020. [Quoted on p. 13, 14]

[22] A. E. P. Sebastian Buettrich, "Basic wireless infrastructure and topologies handout." Obtido de ITRAINONLINE MMTK: `http://www.itrainonline.org/itrainonline/mmtk/wireless_en/04_Infrastructure_Topology/04_en_mmtk_wireless_basic-infrastructure-topology_handout.pdf`. Acedido: 19 de março de 2020. [Quoted on p. 16, 17]

[23] V. Shah, R. Patel, and R. Nayak, "Short range inter-satellite link for data transfer and ranging using ieee802.11n," *International Journal of Computer Applications*, 04 2017. [Quoted on p. 16]

[24] "Networks for iot." Obtido de LPRSTIoT: `http://www.lprsiot.com/networks/`. Acedido: 19 de março de 2020. [Quoted on p. 16, 17]

[25] H. . V. B. . K. I. . M. P. . S. M. Lauridsen, M. ; Nguyen, "Coverage comparison of gprs, nb-iot, lora, andsigfox in a 7800 km2 area," *VTC Spring*, junho de 2017. [Quoted on p. 18]

[26] "Best uses of wireless iot communication technology." Obtido de Industry Today `https://industrytoday.com/article/best-uses-of-wireless-iot-communication-technology/`. Acedido: 16 de novembro de 2019. [Quoted on p. 19, 20, 21, 22, 23]

[27] I. . H. R. Braley, R. ; Gifford, "Wireless personal area networks: An overview of the ieee p802.15 working group, mobile computing and communications review," [Quoted on p. 20]

[28] nControl, "O que é rfid." Obtido de nControl: `https://www.ncontrol.com.pt/o-que-e-rfid.html`. Acedido: 26 de abril de 2020. [Quoted on p. 20]

[29] P. Ahuja, S. ; Potti, "An introduction to rfid technology," *Scientific Research*. [Quoted on p. 20]

[30] Q. . F. T. . L. Q. Jia, X.; Feng, "Rfid technology and its applications in internet of thing (iot)," julho de 2018. [Quoted on p. 20]

[31] "How rfid works." Obtido de Ruddersoft `https://www.ruddersoft.com/blog/how-rfid-works/10`. Acedido: 17 de janeiro de 2020. [Quoted on p. 21]

[32] J. . P. J. Gomez, C. ; Oller, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, agosto de 2012. [Quoted on p. 21]

[33] M. . P. R. Muthu Ramya, C. ; Shanmugaraj, "Study on zigbee technology," *IEEE*, abril de 2011. [Quoted on p. 21]

[34] A. A. H. Mahmoud, M. S. ; Mohamad, "A study of efficient power consumption wireless communication techniques / modules of internet of things (iot) applications," *Scientific Research Publishing (SciRes)*, 22 de abril de 2016. [Quoted on p. 23]

[35] A. Elmangoush, "Evaluating the features of http/2 for the internet of things," 05 2017. [Quoted on p. 25, 28]

[36] S. Ilya Grigorik, "Introdução a http/2." Obtido de Web Fundamentals: `https://developers.google.com/web/fundamentals/performance/http2?hl=pt-br`. Acedido: 6 de julho de 2020. [Quoted on p. 25]

[37] T. M. Tukade and R. Banakar, "Data transfer protocols in iot-an overview," *International Journal of Pure and Applied Mathematics*, vol. 118, pp. 121–138, 01 2018. [Quoted on p. 26]

[38] T. Tukade, "Data transfer protocols in iot- an overview," 02 2018. [Quoted on p. 26, 27]

[39] C. Technology, "Coap overview." Obtido de Coap Technology: `https://coap.technology/`. Acedido: 8 de julho de 2020. [Quoted on p. 26]

[40] L. Tightiz and H. Yang, "A comprehensive review on iot protocols' features in smart grid communication," *Energies*, vol. 13, p. 2762, 06 2020. [Quoted on p. 27]

[41] Noha, N. Omran, A. Ali, and F. Omara, "A survey of internet of things technologies and projects for healthcare services," 2018. [Quoted on p. 30, 31]

[42] V. Nogueira and G. Carnaz, "An overview of iot and healthcare," fevereiro de 2019. [Quoted on p. 30]

[43] M. Dhanvijay and S. Patil, "Internet of things: A survey of enabling technologies in healthcare and its applications," *Computer Networks*, março de 2019. [Quoted on p. 30, 33, 34]

[44] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak, "The internet of things for health care: A comprehensive survey," *IEEE Access*, 2015. [Quoted on p. 32, 33, 34]

[45] C. Hacks, "e-health." Obtido de Cooking Hacks: `https://www.cooking-hacks.com/shop/sensors/e-health.html`. Acedido: 1 de maio de 2020. [Quoted on p. 34]

[46] MySignals, "Mysignals changes the future of medical and ehealth applications." Obtido de MySignals: `http://www.my-signals.com/#buy-mysignals`. Acedido: 1 de maio de 2020. [Quoted on p. 34, 35]

[47] iHealth. Obtido de iHealth: `https://ihealthlabs.com/`. Acedido: 1 de maio de 2020. [Quoted on p. 35]

[48] Semtech. Obtido de Semtech: `https://www.semtech.com/`. Acedido: 14 de agosto de 2020. [Quoted on p. 37]

[49] J. . C. T. . T. W. Augustin, A. ; Yi, "A study of lora: Long range & low power network for the internet of things," *MDPI*, setembro de 2016. [Quoted on p. 37, 38, 40, 44]

[50] "A technical overview of lora and lorawan," *Lora Alliance*, novembro de 2015. [Quoted on p. 38, 44, 45]

[51] A. . C. L. Noreen, U. ; Bouncerr, "A study of lora low power wide area network technology," maio de 2017. [Quoted on p. 38]

[52] T. Bouguera, J.-F. Diouris, J.-J. Chaillout, R. Jaouadi, and A. Guillaume, "Energy consumption model for sensor nodes based on lora and lorawan," *Sensors*, 06 2018. [Quoted on p. 38, 39, 40]

[53] M. B. M. E. H. Ertürk, M.A.; Aydın, "A survey of lorawan for iot: From technology to application," *Future Internet*, 10 2019. [Quoted on p. 38, 41, 44]

[54] A. Augustin, J. Yi, T. H. Clausen, and W. Townsley, "A study of lora: Long range and low power networks for the internet of things," *Sensors*, 10 2016. [Quoted on p. 38, 44, 47]

[55] D.-H. Kim, E.-K. Lee, and J. Kim, "Experiencing lora network establishment on a smart energy campus testbed," *Sustainability*, 03 2019. [Quoted on p. 39, 42]

[56] Semtech, "Sx1276/77/78/79 - 137 mhz to 1020 mhz low power long range transceiver." Obtido de Semtech: `https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276`. Acedido: 15 de maio de 2020. [Quoted on p. 41, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67]

[57] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, "A survey of lorawan for iot: From technology to application," *Sensors*, 11 2018. [Quoted on p. 41, 42]

[58] C. S. P. . B. J. . H. C. . F. J., "Comparison of lorawan classes and their power consumption," novembro de 2017. [Quoted on p. 43, 46]

[59] "Lorawantm 1.1 specification," 11 de outubro de 2017. [Quoted on p. 44, 46, 47]

[60] J. J. . J. R. . A. G. Bouguera T. ; Diouris J. F. ; Chilout, "Energy comsumption model for sensor nodes bade on lora and lorawan," *Sensors*, junho de 2018. [Quoted on p. 46]

[61] "Lorawantm security full end-toend encryption for iot application providers," fevereiro de 2017. [Quoted on p. 47]

[62] Ursalink, "Smart bus tracking." Obtido de Ursalink: `https://www.ursalink.com/en/success-stories/smart-bus-tracking`. Acedido: 28 de abril de 2020. [Quoted on p. 48]

[63] Semtech, "Semtech and x-telia implement lora solution for smart bus schedule signs." Obtido de Semtech: `https://www.semtech.com/company/press/semtech-and-x-telia-implement-lora-solution-for-smart-bus-schedule-signs`. Acedido: 28 de abril de 2020. [Quoted on p. 48]

[64] CHipsafer, "Chipsafer." Obtido de Semtech: `https://www.chipsafer.com/`. Acedido: 28 de abril de 2020. [Quoted on p. 48]

[65] Semtech, "Semtech lora technology and chipsafer connects cattle ranching to the cloud." Obtido de Semtech: `https://www.semtech.com/company/press/semtech-lora-technology-and-chipsafer-connects-cattle-ranching-to-the-cloud`. Acedido: 28 de abril de 2020. [Quoted on p. 48]

[66] Q. AG, "Ear tag for cattle that identifies sick outliers." Obtido de Semtech: `https://quantifiedag.com/`. Acedido: 28 de abril de 2020. [Quoted on p. 49]

[67] Semtech, "Semtech lora™ wireless rf technology selected by quantified ag for smart agriculture applications." Obtido de Semtech: `https://www.semtech.com/company/press/semtech-lora-wireless-rf-technology-selected-by-quantified-ag-for-smart-agriculture-applications`. Acedido: 28 de abril de 2020. [Quoted on p. 49]

[68] Semtech, "Leading the digital transformationof water utilities by enablingautomated and remote meter reading." Obtido de Semtech: `https://info.semtech.com/hubfs/Birdz%20Whitepaper-final_web.pdf`. Acedido: 1 de maio de 2020. [Quoted on p. 49]

[69] "Implementing lora-based solutions for smart metering." Obtido de Semtech: `https://www.allaboutcircuits.com/industry-articles/implemting-lora-based-solution-for-smart-metering-utilities/`. Acedido: 1 de maio de 2020. [Quoted on p. 49]

[70] G. Stream, "Flood sensor." Obtido de Green Stream: `https://greenstream.maxxpotential.org/`. Acedido: 1 de maio de 2020. [Quoted on p. 49]

[71] Lineable. Obtido de Lineable: https://www.lineable.net/. Acedido: 1 de maio de 2020. [Quoted on p. 49]

[72] Careband, "The ultimate technology enhancement for residential dementia care." Obtido de Careband: https://www.carebandremembers.com/#overview. Acedido: 1 de maio de 2020. [Quoted on p. 49]

[73] Semtech, "Fall detection." Obtido de Semtech: https://www.semtech.com/uploads/technology/LoRa/app-briefs/AB-SEMTECH-LORA-SMART-HEALTH-FALL-DETECTION.PDF. Acedido: 1 de maio de 2020. [Quoted on p. 50]

[74] Semtech, "Medical refrigerators." Obtido de Semtech: https://www.semtech.com/uploads/technology/LoRa/app-briefs/Semtech_Health_MedicalFridge_AppBrief-FINAL.pdf. Acedido: 1 de maio de 2020. [Quoted on p. 50]

[75] "Lora alliance." Obtido de LoRa Alliance: https://lora-alliance.org/. Acedido: 2 de abril de 2020. [Quoted on p. 50]

[76] "Semtech lora products." Obtido de Semtech: https://www.semtech.com/lora/lora-products. Acedido: 3 de abril de 2020. [Quoted on p. 50, 51, 52, 53]

[77] "Hoperf lora module." Obtido de HopeRF: https://www.hoperf.com/modules/lora/index.html. Acedido: 5 de abril de 2020. [Quoted on p. 53]

[78] "Microchip lora technology." Obtido de Microchip: https://www.microchip.com/design-centers/wireless-connectivity/low-power-wide-area-networks/lora-technology. Acedido: 5 de abril de 2020. [Quoted on p. 53]

[79] "Wemos ttgo lora32." Obtido de BotnRoll: https://www.botnroll.com/pt/varias/3386-wemos-ttgo-lora32-868mhz-sx1276-c-esp32-wifi-bluetooth-e-display-oled.html. Acedido: 7 de abril de 2020. [Quoted on p. 54]

[80] "Pycom lopy4." Obtido de Pycom: https://pycom.io/product/lopy4/. Acedido: 8 de abril de 2020. [Quoted on p. 54, 55]

[81] "Lora shield for arduino." Obtido de Dragino: https://www.dragino.com/products/module/item/102-lora-shield.html. Acedido: 8 de abril de 2020. [Quoted on p. 55]

[82] "Semtech sx1276mb1mas." Obtido de Mouser: https://pt.mouser.com/ProductDetail/Semtech/SX1276MB1MAS?qs=%2FukktrhNKsauG%252Bdxd7cK4A==. Acedido: 8 de abril de 2020. [Quoted on p. 56]

[83] "Dragino lora bee." Obtido de Dragino: https://www.dragino.com/products/lora/item/109-lora-bee.html. Acedido: 9 de abril de 2020. [Quoted on p. 56]

[84] "Adafruit rfm95w lora radio transceiver." Obtido de Adafruit: https://www.adafruit.com/product/3072. Acedido: 9 de abril de 2020. [Quoted on p. 56]

[85] "Rfm95w lora module." Obtido de HopeRF: https://www.hoperf.com/modules/lora/RFM95.html. Acedido: 9 de abril de 2020. [Quoted on p. 56]

[86] Atmel, "Atmega88/atmega168 datasheet." Obtido de Microchip: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-9365-Automotive-Microcontrollers-ATmega88-ATmega168_Datasheet.pdf. Acedido: 13 de agosto de 2020. [Quoted on p. 71]

[87] Arduino, "Pinmapping168." Obtido de Arduino: https://www.arduino.cc/en/Hacking/PinMapping168. Acedido: 13 de agosto de 2020. [Quoted on p. 71]

[88] Microchip, "Atmel studio 7." Obtido de Microchip: https://www.microchip.com/mplab/avr-support/atmel-studio-7. Acedido: 13 de agosto de 2020. [Quoted on p. 72]

[89] Components101, "Nodemcu esp8266." Obtido de Components101: https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet. Acedido: 14 de agosto de 2020. [Quoted on p. 72, 73]

[90] E. Systems, "Esp8266ex datasheet." Obtido de Expressif: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Acedido: 14 de agosto de 2020. [Quoted on p. 72]

[91] R. Solutions, "Longrange transceiver." Obtido de RF Solutions: https://www.rfsolutions.co.uk/downloads/1594122930DS-RFLoRa-7.pdf. Acedido: 14 de agosto de 2020. [Quoted on p. 73, 74]

[92] Semtech, "Sx1272/73 - 860 mhz to 1020 mhz low power long range transceiver." Obtido de Semtech: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1272. Acedido: 14 de agosto de 2020. [Quoted on p. 73, 113]

[93] D. Soni and A. Makwana, "A survey on mqtt: A protocol of internet of things (iot)," 04 2017. [Quoted on p. 75]

[94] E. C. Ivan Vaccari, Maurizio Aiello, "Slowite, a novel denial of service attack affecting mqtt," *Sensors 2020*, 05 2020. [Quoted on p. 75]

[95] NodeJS, "About nodejs." Obtido de NodeJS: `https://nodejs.org/en/about/`. Acedido: 15 de agosto de 2020. [Quoted on p. 76]

[96] TutorialsPoint, "Node.js - express framework." Obtido de TutorialsPoint: `https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm`. Acedido: 15 de agosto de 2020. [Quoted on p. 76]

[97] TutorialsPoint, "Mongodb - overview." Obtido de TutorialsPoint: `https://www.tutorialspoint.com/mongodb/mongodb_overview.htm`. Acedido: 15 de agosto de 2020. [Quoted on p. 76]

[98] V. S. Code. Obtido de Visual Studio Code: `https://code.visualstudio.com/`. Acedido: 16 de agosto de 2020. [Quoted on p. 77]

[99] ReactJS. Obtido de ReactJS: `https://reactjs.org/`. Acedido: 17 de agosto de 2020. [Quoted on p. 77]

[100] CEPT-ECC, "Erc recommendation 70-03." Obtido de CEPT: `https://docdb.cept.org/download/25c41779-cd6e/Rec7003e.pdf`. Acedido: 10 de outubro de 2020. [Quoted on p. 118]

[101] T. T. Network, "Lorawan duty cycle." Obtido de The Things Network: `https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html`. Acedido: 10 de outubro de 2020. [Quoted on p. 118]

[102] ETSI, "European standard." Obtido de ETSI: `https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/02.04.01_40/en_30022001v020401o.pdf`. Acedido: 10 de outubro de 2020. [Quoted on p. 118]

Anexo A

Mapa de Memória SX1276

Tabela 1 : Mapa de memória SX1276/77/78/79

Nome (endereço)	Bits	Nome da variável	*Reset*	Descrição
Registos Comuns				
RegFifo (0x00)	7-0	Fifo	0x00	Registo de dados FIFO
RegOpMode (0x01)	7	LongRangeMode	0x0	Modos de funcionamento: 0 – FSK/OOK 1 - LoRa
	2-0	Mode	0x01	Modos de operação: 000 - Sleep 001 - Standby 011 - TX 101 - RxContinuous 110 - RxSingle
RegFrMsb (0x06)	7-0	Frf(23:16)	0x6c	Frequência Portadora $f_{RF} = (F_{XOSC} * F_{rf}) / 2^{19}$
RegFrMid (0x07)	7-0	Frf(15:8)	0x80	
RegFrLsb (0x08)	7-0	Frf(7:0)	0x00	
Registos de amplificação do módulo RF				
RegPaConfig (0x09)	7	PaSelect	0x00	Seleciona o PA de saída: 0 →RFO pin. Potência de saída limitada até +14 dBm 1 →PA_BOOST pin. Potência de saída limitada até +20 dBm
	6-4	MaxPower	0x04	Pmax=10.8+0.6*MaxPower [dBm]
	3-0	OutputPower	0x0f	Pout=Pmax-(15-OutputPower) , caso RFO pin. Pout=17-(15-OutputPower) , caso PA_BOOST pin
RegPaRamp (0x0A)	3-0		0x09	Estabelecimento do tempo de subida/descida no modo FSK
RegOcp (0x0B)	5	OcpOn	0x01	Ativa OCP (Overload Current Protection) para os PA 0 → OCP disabled ; 1 → OCP enabled
	4-0	OcpTrim	0x0b	Valor da corrente para o corte através OCP
RegLna (0x0C)	7-5	LnaGain	0x01	Definição do ganho. De G1 ‘001’ a G6 ‘110’
	1-0	LnaBoostHf	0x00	00 → Corrente nos LNA default 11 → Boost ativado, corrente nos LNA a 150 %
Registos do buffer de dados FIFO				
RegFifoAddrPtr (0x0D)	7-0	FifoAddrPtr	0x00	Apontador do endereço do buffer de dados FIFO para a interface SPI
RegFifoTxBaseAddr (0x0E)	7-0	RegFifoTxBaseAddr	0x80	Endereço base do buffer de dados FIFO utilizado para armazenar os dados a enviar
RegFifoRxBaseAddr (0x0F)	7-0	RegFifoRxBaseAddr	0x00	Endereço base do buffer de dados FIFO utilizado para armazenar os dados a receber
RegFifoRxCurrentAddr (0x10)	7-0	RegFifoRxCurrentAddr	-	Endereço do último pacote recebido no buffer de dados FIFO
RegFifoRxByteAddr (0x25)	7-0	FifoRxByteAddrPtr	-	Endereço do pacote que se encontra a receber no momento
RegRxNbBytes (0x13)	7-0	FifoRxBytesNb	-	Indica o número de bytes recebidos no último pacote de dados

Registos relativos a interrupções e pinos I/O				
RegIrqFlagsMask (0x11)	7	RxTimeoutMask	0x00	Registo utilizado para ativar/desativar interrupções. Um bit a 1 indica que a interrupção se encontra desativada.
	6	RxDoneMask	0x00	
	5	PayloadCrcErrorMask	0x00	
	4	ValidHeaderMask	0x00	
	3	TxDoneMask	0x00	
	2	CadDoneMask	0x00	
	1	FhssChangeChannelMask	0x00	
	0	CadDetectedMask	0x00	
RegIrqFlags (0x12)	7	RxTimeout	0x00	Registo utilizado para verificar o estado das interrupções. Um bit a 1 indica que a respetiva interrupção ocorreu.
	6	RxDone	0x00	
	5	PayloadCrcError	0x00	
	4	ValidHeader	0x00	
	3	TxDone	0x00	
	2	CadDone	0x00	
	1	FhssChangeChannel	0x00	
	0	CadDetected	0x00	
RegDioMapping1 (0x40)	7-6	DIO0Mapping	0x00	Mapeamento dos pinos DIO0 – DIO03 Consultar Tabela 3.12 (LoRa)
	5-4	DIO1Mapping	0x00	
	3-2	DIO2Mapping	0x00	
	1-0	DIO3Mapping	0x00	
RegDioMapping2 (0x41)	7-6	DIO4Mapping	0x00	Mapeamentos dos pinos DIO4 e DIO05 Consultar Tabela 3.12 (LoRa)
	5-4	DIO5Mapping	0x00	
Registos relativos aos parâmetros de modulação e pacotes LoRa				
RegModemConfig1 (0x1D)	7-4	BW	0x07	Bandwidth: 0111 – 125 kHz 1000 – 250 kHz 1001 – 500 kHz
	3-1	CodingRate	'001'	CodingRate: 001 – 4/5 010 – 4/6 011 – 4/7 100 – 4/8
	0	ImplicitHeaderModeOn	0x0	0 – Header explícito 1 – Header implícito
RegModemConfig2 (0x1E)	7-4	SpreadingFactor	0x7	SpreadingFactor: 0111 – SF7 1000 – SF8 1001 – SF9 1010 – SF10 1011 – SF11 1100 - SF12
	3	TxContinuousMode	0	0 – Modo normal 1 – Modo contínuo
	2	RxPayloadCrcOn	0x00	0 – CRC desativado 1 – CRC ativado
	1-0	SymbTimeout(9:8)	0x00	Tamanho do Timeout: *TimeOut = SymbTimeout * Ts*
RegSymbTimeoutLsb (0x1F)	7-0	SymbTimeout(7:0)	0x64	
RegPreambleMsb (0x20)	7-0	PreambleLength (15:8)	0x0	Tamanho do preamble
RegPreambleLsb (0x21)	7-0	PreambleLength (7:0)	0x8	
RegPayloadLength (0x22)	7-0	PayloadLength(7:0)	0x1	Tamanho do payload em bytes

Anexo B

Código Desenvolvido para o Periférico SPI

```
void SelectReceiver(){
  // Enable CS (CS --> LOW)
  PORTB &= ~(1 << CS);
}

void UnselectReceiver(){
  // Disable CS (CS --> HIGH)
  PORTB |= (1 << CS);
}

uint8_t SPI_WriteRegister(uint8_t addr, uint8_t data){
   // transmit data
   SPDR = addr;
   // Wait for reception complete
   while(!(SPSR & (1 << SPIF)));
   // transmit dummy byte
   SPDR = data;
   // Wait for reception complete
   while(!(SPSR & (1 << SPIF)));
   // return Data Register
   return SPDR;
}
```

Anexo C

Módulo LoRa - Funções e Configurações

```
void LoRa_WriteRegister(uint8_t RegAddr, uint8_t data){
  SelectReceiver();
  SPI_WriteRegister((RegAddr | 0x80), data);
  UnselectReceiver();
}

void Switch_TX (){
  PORTD |= (1<<TX_SWITCH);
  PORTD &= ~(1<<RX_SWITCH);
}

void Switch_RX(){
  PORTD |= (1<<RX_SWITCH);
  PORTD &= ~(1<<TX_SWITCH);
}

void Op_Mode(uint8_t Mode){
  LoRa_WriteRegister(REG_OP_MODE, ((LoRa_ReadRegister(REG_OP_MODE) &
          (OP_MODE_MASK)) | Mode));
}
```

Tabela C.1: Configurações efetuadas no módulo de rádio LoRa.

Parâmetros de modulação LoRa
Carrier Frequency 869.500 MHz
BW 125 kHz
CR 4/8
Header Explicit
CRC On
Low Data Rate Optimization Ativado (Obrigatório na utilização de SF11 e SF 12 com BW 125 kHz)
SF11
Tx continuous mode Off (default)
Preamble Length 12,25 Symbols (default)
Payload Length 10 bytes (Apenas necessário caso seja utilizado header implícito)
Max payload Length 255 bytes (default)
Lora detection optimized for SF7 to SF12
Lora detection threshold to SF7 and SF12
Parâmetros de amplificação
PA output pin PA_BOOST (Aumenta o limite de amplificação até 20 dBm)
Low consumption PLL is used in receive and transmit mode (default)
Overload current protection (OCP) for PA enable (default)
LNA maximum gain and Boost on (default)
+20 dBm option on PA_BOOST pin Enable

```
void LoRa_Config(){
  uint8_t version = LoRa_ReadRegister(REG_VERSION);
  if (version == REG_VERSION_VALUE){

    // Change to Sleep mode for configuration
    Op_Mode(OP_MODE_SLEEP);

    // Set modem LoRa
    LoRa_WriteRegister(REG_OP_MODE, (LoRa_ReadRegister(REG_OP_MODE) | OP_MODE_LORA));

    // Set standby mode
    Op_Mode(OP_MODE_STANDBY);

    //Set Carrier Frequency (based on datasheet pag. 109)
    // frf => 0xD90000 for 868 MHz     // frf => 0xD96080 for 869.500 MHz
    LoRa_WriteRegister( REG_FR_MSB, 0xD9 );
    LoRa_WriteRegister( REG_FR_MID, 0x60 );
    LoRa_WriteRegister( REG_FR_LSB, 0x80 );

    // Set PA_BOOST pin
    LoRa_WriteRegister(REG_PA_CONFIG, 0x8F);
    // Low comsumption PLL is used in receive and transmit mode
    LoRa_WriteRegister(REG_PA_RAMP, 0x19);
    // OCP enable
    LoRa_WriteRegister(REG_OCP, 0x2B);
```

```
// G1 maximum gain and Boost on, 150% LNA current
LoRa_WriteRegister(REG_LNA, 0x23);

// Set BW --> BW 125 MHz
LoRa_WriteRegister(REG_MODEM_CONFIG1, ((LoRa_ReadRegister(REG_MODEM_CONFIG1) & MC1_BW_MASK) | BW));
// Set CR --> CR4_5
LoRa_WriteRegister(REG_MODEM_CONFIG1, ((LoRa_ReadRegister(REG_MODEM_CONFIG1) & MC1_CR_MASK) | CR));
// Set Header moder --> Explicit header
LoRa_WriteRegister(REG_MODEM_CONFIG1, ((LoRa_ReadRegister(REG_MODEM_CONFIG1) & MC1_IMPLICIT_HEADER_MASK)
                    | MC1_IMPLICIT_HEADER_MODE_OFF_EXPLICIT));
// Set CRC --> CRC on
LoRa_WriteRegister(REG_MODEM_CONFIG1, ((LoRa_ReadRegister(REG_MODEM_CONFIG1) & MC1_RX_PAYLOAD_CRC_MASK)
                    | MC1_RX_PAYLOAD_CRC_ON));
// Set Low Data Rate Optimization --> Low data rate optimize on
LoRa_WriteRegister(REG_MODEM_CONFIG1, ((LoRa_ReadRegister(REG_MODEM_CONFIG1)
                   & MC1_LOW_DATA_RATE_OPTIMIZE_MASK) | MC1_LOW_DATA_RATE_OPTIMIZE_ON));
// Set SF --> SF11
LoRa_WriteRegister(REG_MODEM_CONFIG2, ((LoRa_ReadRegister(REG_MODEM_CONFIG2) & MC2_SF_MASK) | SF));
// Set Tx continuous mode On --> Tx continuous mode off
LoRa_WriteRegister(REG_MODEM_CONFIG2, ((LoRa_ReadRegister(REG_MODEM_CONFIG2) & MC2_TX_CONTINUOUS_MODE_MASK)
                    | MC2_TX_CONTINUOUS_MODE_OFF));
// Set AcgAutoOn --> AcgAutoOn on
LoRa_WriteRegister(REG_MODEM_CONFIG2, ((LoRa_ReadRegister(REG_MODEM_CONFIG2) & MC2_AGC_AUTO_ON_MASK)
                    | MC2_AGC_AUTO_ON_ON));
```

```
// Receiver Timeout (Max time of Rx_Timeout Interrupt)   // 0xB7 => Timeout value for SF11 and BW125 => 3 s
// 0x06 => Timeout value for SF11 and BW125 => 98.4 ms  // 0x05 => Timeout value for SF11 and BW125 => 82 ms
// 0x04 => Timeout value for SF11 and BW125 => 65.4 ms
LoRa_WriteRegister(REG_SYMB_TIMEOUT_LSB, 0x05);

// Preamble MSB and LSB --> 6 (+ 4.25)
LoRa_WriteRegister(REG_PREAMBLE_MSB, 0x00);
LoRa_WriteRegister(REG_PREAMBLE_LSB, 0x06);

// Payload length
LoRa_WriteRegister(REG_PAYLOAD_LENGTH, 10);
// Max payload
LoRa_WriteRegister(REG_MAX_PAYLOAD_LENGTH, 0xFF);

// Hop Period (Minimum hop period)
LoRa_WriteRegister(REG_HOP_PERIOD, 0x01);
// Lora detection optimized for SF7 to SF12
LoRa_WriteRegister(REG_DETECT_OPTIMIZE, 0x03);
// Lora detection threshold to SF7 and SF12
LoRa_WriteRegister(REG_DETECTION_THRESOLD, 0x0A);

//High power PA control (set high power)
LoRa_WriteRegister(REG_PA_DAC, 0x87);
```

```
    sprintf(transmit_buffer,"\r LoRa config completed! \n\n");
    Usart_transmit_message(transmit_buffer);
  }
  else{
    sprintf(transmit_buffer,"\r Error! Connection with SX1272 failed! \n");
    Usart_transmit_message(transmit_buffer);
  }
}
```

INSTITUTO SUPERIOR DE

isep

Sistema de Comunicação baseado na Tecnologia LoRa para

FLAVIO PEREIRA BRANCO

isep Instituto Superior de Engenharia do Porto

Instituto Superior de Engenharia do Porto ⊙ Rua Dr. António Bernardino de Almeida, 431 ⊙ 4249-015 Porto
Tel. +351 228 340 500 ⊙ Fax +351 228 321 159 ⊙ mail@isep.ipp.pt ⊙ www.isep.ipp.pt