INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Plataforma de Controlo e Comando para Instalações Artísticas

Tiago Alexandre Mendes Domingos

(Licenciado)

Trabalho de Projeto para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Orientador : Prof. Doutor Nuno Miguel Machado Cruz

Júri:

Presidente: Prof. Nuno Cota

Vogal: Prof. Doutor João Ferreira

Junho, 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Plataforma de Controlo e Comando para Instalações Artísticas

Tiago Alexandre Mendes Domingos

(Licenciado)

Trabalho de Projeto para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Orientador : Prof. Doutor Nuno Miguel Machado Cruz

Júri:

Presidente: Prof. Nuno Cota

Vogal: Prof. Doutor João Ferreira

Junho, 2021

Agradecimentos

Antes demais, quero agradecer ao Professor Nuno Cruz que me ajudou ao longo do desenvolvimento deste projeto, especialmente com a estratégia e metodologia a implementar para a obtenção dos melhores resultados possíveis.

Aos meus amigos do ISEL, André Rouco, Pedro Prior, Paulo Capitão, David Rodrigues e Gonçalo Costa pela entreajuda demonstrada nos mais diversos tópicos relacionados com o trabalho e pela grande amizade cimentada durante estes anos.

À minha namorada, Marta Pinto, pela ajuda neste projeto e também por toda a paciência e compreensão que teve para comigo durante os períodos de maior trabalho.

Por fim, mas não menos importante, quero agradecer aos pais, avós e irmão por me terem apoiado em todo o meu percurso até aos dias de hoje.

Resumo

Atualmente, com o aumento de tecnologias e protocolos associados à *Internet of Things* (IoT), assistimos a uma procura cada vez maior de sistemas dotados de automatismos ou controlados e geridos à distância, seja através do envio de comandos ou mediante a interação entre as varias partes que constituem o sistema.

Neste contexto, a Musa Paradisíaca no decorrer das suas atividades artísticas identificou uma lacuna importante: A falta de uma plataforma de operacionalidade simples e de baixo custo que permita o controlo e o comando dos diversos elementos que compõem a sua instalação artística.

Neste projeto foi desenvolvido e implementado uma plataforma para o controlo e gestão de uma rede de dispositivos. A interface com a plataforma foi conseguida através do desenvolvimento de uma aplicação *web* que permite a gestão dos diversos elementos assim como o envio de instruções para o controlo do fluxo de ações da rede de dispositivos (de acordo com o cenário idealizado pelo utilizador).

A comunicação com a rede de dispositivos é estabelecida através do protocolo *Long Range Wide Area Network* (LoRaWAN) com o suporte da infraestrutura de rede IoT - *The Things Network* (TTN). A interação entre os dispositivos da rede é assegurada através da implementação do protocolo *ESP-Mesh*. Cada dispositivo, integrado com microcontrolador *ESP32*, está também a capacitado para interagir com sensores e atuadores.

Foi também desenvolvida uma linguagem onde se definiu o formato de todo o tipo de informação trocada entre a aplicação e a rede de dispositivos e entre os próprios dispositivos. No âmbito desta linguagem, foram também elaborados vários comandos que podem ser aplicados pelo utilizador com o objetivo de programar o comportamento da rede de dispositivos. A diversidade de comandos implementada permite a adaptação desta plataforma aos mais variados cenários relacionados com a interação entre dispositivos, sensores e atuadores.

Palavras-chave: *Internet of Things*, *LoRaWAN*, *LoRa*, *The Things Network*, *ESP32*, *ESP-Mesh*, Plataforma de controlo e gestão, Linguagem de programação, Comandos

Abstract

Nowadays, with the increase of technologies and protocols associated with IoT, we are witnessing an increasing demand for systems equipped with automation or controlled and managed remotely, either by sending commands or through interaction between the various parts that constitute the system.

In this context, the Musa Paradisíaca in the course of its artistic activities identified an important gap: the lack of a simple and low-cost operating platform that allows the control and command of the various elements that compose its artistic installation.

In this project was developed and implemented a platform for the control and management of a devices network. The interface with the platform was achieved through the development of a web application that allows the management of the various elements as well as sending instructions for controlling the flow of actions of the network devices (according to the scenario idealized by the user).

The communication with the network of devices is established through the LoRaWAN protocol with the support of the IoT network infrastructure - TTN. The interaction between the devices in the network is ensured by implementing the ESP-Mesh protocol. Each device, integrated with ESP32 micro-controller, is also capable of interacting with sensors and actuators.

A language was also developed where the format of all types of information exchanged between the application and the network of devices and between the devices themselves was defined. Within the scope of this language, various commands were also developed that can be applied by the user with the aim of programming the behaviour of the network of devices. The diversity of commands implemented allows the adaptation of this platform to the most varied scenarios related to the interaction between devices, sensors and actuators.

Keywords: Internet of Things, LoRaWAN, LoRa, The Things Network, ESP32, ESP-Mesh, Platform for control and management, Programming language, Commands

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

3GPP	*3rd Generation Partnership Project*
AP	*Access Point*
BLE	*Bluetooth Low Energy*
BPSK	*Binary Phase-Shift Keying*
CSS	*Chirp Spread Spectrum*
EC-GSM	*Extended Coverage GSM*
GSM	*Global System for Mobile Communications*
GUI	*Graphical User Interface*
IoT	*Internet of Things*
LoRa	*Long Range*
LoRaWAN	*Long Range Wide Area Network*
LPWAN	*Low-Power Wide-Area Network*
LTE	*Long-Term Evolution*
LTE-M	*Long Term Evolution Machine*
MAC	*Media Access Control*
MIDI	*Musical Instrument Digital Interface*
NB-IoT	*NarrowBand-Internet of Things*
RSSI	*Received Signal Strength Indication*
SF	*Spreading Factor*
SIG	*Special Interest Group*
TOA	*Time On Air*
TTN	*The Things Network*
UNB	*Ultra Narrow-Band*
WLAN	*Wireless LAN*

1

Introdução

1.1 Enquadramento

O desenvolvimento deste projeto surge de uma ideia da Musa Paradisíaca no sentido de ultrapassar um obstáculo encontrado no decorrer das suas atividades artísticas. Musa Paradisíaca é um projeto artístico de Eduardo Guerra e Miguel Ferrão. Iniciada em 2010, Musa Paradisíaca tem produzido esculturas, filmes e desenhos assim como sessões ou ações performativas.

Na perspectiva teatral inerente à prática artística de Musa Paradisíaca existe a necessidade de conjugar os múltiplos objetos artísticos fisicamente no mesmo espaço e na mesma linha temporal. O meio de interação pode ser, por exemplo, sob a forma de iluminação, sensores de reação ou atuadores físicos. Face à situação exposta surgiu a necessidade de encontrar uma plataforma de simples interação para o controlo e comando dos vários elementos artísticos que habitualmente compõem as ações performativas realizadas pela Musa Paradisíaca.

1.2 Motivação

Tendo em conta o problema exposto pela Musa Paradisíaca o desenvolvimento e implementação de uma plataforma que permita a sincronização temporal de múltiplos

elementos implica a criação de um sistema de relações versátil, mutável, de fácil operação em contextos de apresentação múltiplos e escalável a diferentes situações e tipologias de açao. Entre estas ações principais prevê-se, por exemplo, a sincronização entre luz e objectos, em que a iluminação é activada de acordo com uma linha de tempo pré-determinada ou através da reação a um estímulo externo que a inicie (sensor de movimento, toque ou outro).

Apesar de existir alguma oferta que soluciona o problema colocado pela Musa Paradisíaca, todos apresentam um elevado grau de complexidade, custo de aquisição e manutenção. Interessa, por isso mesmo, colocar sob hipótese a construção de um sistema modular de baixo custo e de operacionalidade acessível para colocar, no âmbito desta pratica artística, em acção um principio de atribuir sensibilidade às coisas e a possibilidade de adaptação a outros cenários.

O desenvolvimento desta plataforma, tendo em conta a forma como a mesma será implementada, permite a adaptação aos mais variados cenários relacionados com a interação entre dispositivos e sensores, através de uma plataforma de controlo e gestão destes elementos.

1.3 Objetivos

O desafio colocado pela Musa Paradisíaca consiste, de forma genérica, em desenvolver um sistema de **interface**, **controlo** e **gestão** para melhorar e facilitar as suas atividades conseguindo, desta forma, aumentar o impacto da sua expressão artística.

Do ponto de vista tecnológico, pretende-se criar uma plataforma (controlada externamente) composta por múltiplos dispositivos remotos que comuniquem entre si de forma a sincronizar vários estímulos induzindo diferentes comportamentos, criando assim fluxos de ações.

Para o desenvolvimento e implementação desta plataforma, após a análise da proposta, é necessário ter em consideração os seguintes requisitos:

- A **gestão** da plataforma deve ser assegurada através da implementação e utilização de uma infraestrutura de rede IoT, responsável por assegurar a comunicação entre a aplicação *Web* e a rede de dispositivos.
  - Utilização de uma tecnologia de comunicação de longa distancia para interface entre a aplicação desenvolvida e a rede de dispositivos.

- A **interface** com a plataforma deverá ser conseguida através do desenvolvimento de uma aplicação *Web*, onde o utilizador tenha a possibilidade de gerir os diversos dispositivos assim como controlar fluxo de ações executadas pela rede de dispositivos.
  - Desenvolvimento de uma aplicação de gestão da plataforma que permita a configuração dos fluxos de ações dos vários elementos do sistema.
- O **controlo** da plataforma deve ser conseguido através da implementação e desenvolvimento de uma rede *Mesh*, responsável pelo sincronismo de fluxos de ações entre os diversos dispositivos.
  - Os dispositivos que compõem o sistema têm de ter a capacidade de interagir com os mais variados tipos de sensores (por exemplo sensores de movimento, pressão ou temperatura) e de atuar diretamente sobre o objecto artístico em que estão instalados.
  - Utilização de micro-controladores de baixo custo.
  - O sincronismo de fluxos de ações entre os diferentes dispositivos será através da exploração das capacidades de redes *Mesh*, tendo sempre em conta que, dado o ambiente artístico em causa, é fundamental precaver a necessidade de comunicações a distâncias na ordem das dezenas ou centenas de metros.
- Otimização energética da solução desenvolvida é importante mas não é um fator limitativo uma vez que o ambiente onde o sistema será implementado está munido de fonte de alimentação.

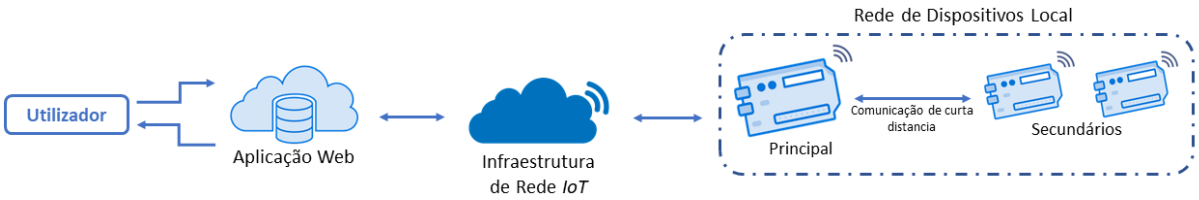


Figura 1.1: Arquitetura da solução

Na figura 1.1 está representada a arquitectura do sistema a implementar para mitigar a lacuna exposta pela Musa Paradisíaca.

A solução desenvolvida está dividida em três setores:

- **Aplicação *Web*:** É através da aplicação que o utilizador faz a gestão e o controlo do seu sistema. Visualiza o estado da rede e dos seus elementos e envia instruções para a rede de dispositivos executar.

- **Rede Local:** Composta por um dispositivo Principal e vários dispositivos Secundários. O dispositivo Principal é responsável pela recolha e tratamento da informação transmitida entre a aplicação e a instalação artística. Os dispositivos Secundários são os elementos ativos da instalação artística, estão ligados a sensores e atuadores tornando possível a execução das instruções provenientes do dispositivo Principal.
- **Infraestrutura de Rede:** Plataforma de suporte e gestão da comunicação de longa distância, responsável por assegurar a interface entre a aplicação *Web* e a rede de dispositivos.

1.4 Organização do documento

Este documento está organizado da seguinte forma. O **Capítulo 1** detalha o enquadramento, motivação e objetivos deste projeto. No **Capítulo 2** é apresentado o estado da arte com referência a tecnologias *Low-Power Wide-Area Network* (LPWAN), infraestruturas de rede IoT, ferramentas de programação, micro controladores e protocolos de comunicação de curta distância, que são fundamentais para o desenvolvimento deste projeto. O **Capítulo 3** apresenta a metodologia utilizada no desenvolvimento e implementação deste projeto, através do detalhe da arquitetura física, com a descrição dos micro-controladores utilizados, e de a arquitetura lógica com a descrição da linguagem desenvolvida, aplicação de gestão e controlo da rede *mesh*, configurações da infraestrutura de rede IoT e configurações da rede *mesh*. O **Capítulo 4** apresenta as conclusões do trabalho desenvolvido e algumas propostas de trabalho futuro.

2

Estado da arte

Neste capítulo são descritas e enumeradas diversas alternativas tecnológicas às utilizadas neste projecto, tendo como principal objectivo de expor o raciocínio que deu origem às escolhas tomadas nas diversas fases do projecto, para a implementação e integração da solução final.

2.1 Tecnologias *LPWAN*

A IoT baseia-se na interligação e transmissão de dados entre dispositivos/sensores. As aplicações da IoT têm requisitos específicos tais como, baixo ritmo de transmissão de dados, baixo consumo de energia e eficácia de custos. As tecnologias de rádio de curto alcance amplamente utilizadas (por exemplo, *ZigBee* ou *Bluetooth*) não são adaptadas para cenários que requerem transmissão de longo alcance. As soluções baseadas em comunicações celulares (por exemplo, 2G, 3G, e 4G) podem proporcionar uma melhor cobertura, mas apresentam pouca eficiência energética e elevados custos de manutenção. Por conseguinte, os requisitos das aplicações IoT têm impulsionado o surgimento de uma nova tecnologia de comunicação: LPWAN [1].

Atualmente estão disponíveis, para a IoT, diversas soluções tecnológicas para comunicações de longa distância. É possível dividir estas tecnologias em 2 grandes grupos, as que operam em banda não licenciada e as que disponibilizam o seu serviço em banda licenciada.

Tecnologias que operam em banda não licenciada selecionadas para análise:

- LoRa/LoRaWAN;
- *SigFox*.

Tecnologias que operam em banda licenciada selecionadas para análise:

- *Long Term Evolution Machine* (LTE-M);
- *NarrowBand-Internet of Things* (NB-IoT);
- *Extended Coverage GSM* (EC-GSM).

2.1.1 LoRa e LoRaWAN

Ao abordar esta tecnologia, é necessário distinguir os dois conceitos inerentes à mesma: LoRa e LoRaWAN. LoRa é uma tecnologia de modulação baseada na técnica de espalhamento na frequência (*Chirp Spread Spectrum* (CSS)). LoRaWAN é o nome dado ao protocolo que define os parâmetros de comunicação usando a tecnologia LoRa para criar uma rede. LoRa é apenas a componente física (modulação e camada física de transporte de bits), tudo o que diz respeito à gestão de frequências, endereços, capacidades de transmissão/receção e camada *Media Access Control* (MAC) é LoRaWAN.

Esta tecnologia utiliza banda não licenciada e é totalmente livre, ou seja, qualquer pessoa tendo adquirido um dispositivo com capacidade de transmitir com a modulação LoRa pode usufruir da rede. Para uma ligação ponto-a-ponto (entre 2 dispositivos LoRa) não é obrigatório utilizar LoRaWAN a modulação LoRa é suficiente, mas se a aplicação desenvolvida necessitar de utilizar a rede global neste caso já é necessário usar a componente de LoRaWAN.

Um dos aspectos mais importantes nesta tecnologia é a modulação nela implementada, o CSS, ou seja, o espalhamento na frequência utilizando um *chirp*.

Chirp (figura 2.1) é uma forma de onda que varia a sua frequência ao longo do tempo, começa com uma baixa frequência e passa para uma alta frequência (*up-chirp*) ou o contrário, passa de uma alta frequência para uma baixa frequência (down-chirp). O princípio de espalhamento do sinal é simples, cada símbolo do sinal original é multiplicado pelo *chirp* espalhando, desta forma, o sinal original pela frequência. O resultado desta operação é passar um sinal de frequência fixa, para um sinal espalhado em toda a banda do canal. O que possibilita a comunicação entre dispositivos sem sincronismo e torna a comunicação imune ao efeito de *Doppler* [2].

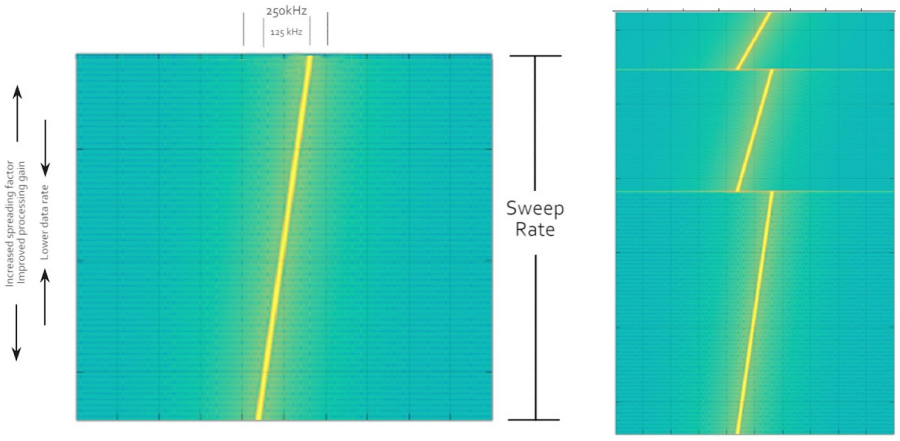


Figura 2.1: *LoRa Chirp*

Esta é a grande inovação do LoRa, transmitir diferentes símbolos através da variação de frequência. A duração de um *chirp* (tempo total da variação de frequência) é um símbolo LoRa. A distinção entre os diferentes símbolos LoRa é feita através da diferença entre a frequência inicial e final do *chirp*. O recetor LoRa deteta a chegada de uma mensagem como sendo um pacote, quando recebe oito *up-chirps* seguidos de dois *down-chirps* como demonstrado na figura 2.2 [2].

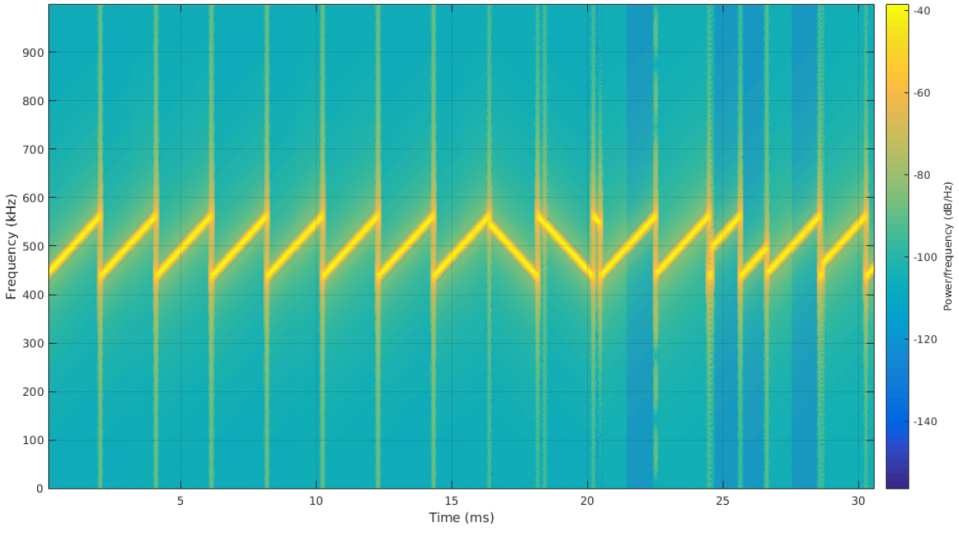


Figura 2.2: Pacote LoRa

Outro aspecto a ter em consideração na utilização de LoRa/LoRaWAN num projecto é o *Spreading Factor* (SF), uma vez que este parâmetro tem um impacto considerável no consumo energético, alcance máximo e número máximo de mensagens transmitidas. LoRa utiliza seis SFs programáveis que vão desde SF7 a SF12.

Um símbolo LoRa contém SF bits, ou seja, um aumento do SF produz um aumento no tempo de símbolo e um aumento da energia do símbolo o que resulta num maior alcance, mas em contrapartida produz também um aumento do consumo energético e um aumento dos recursos rádio utilizados. Este efeito pode ser observado na figura 2.3 onde está representado o *Time On Air* (TOA) para os diferentes SFs.

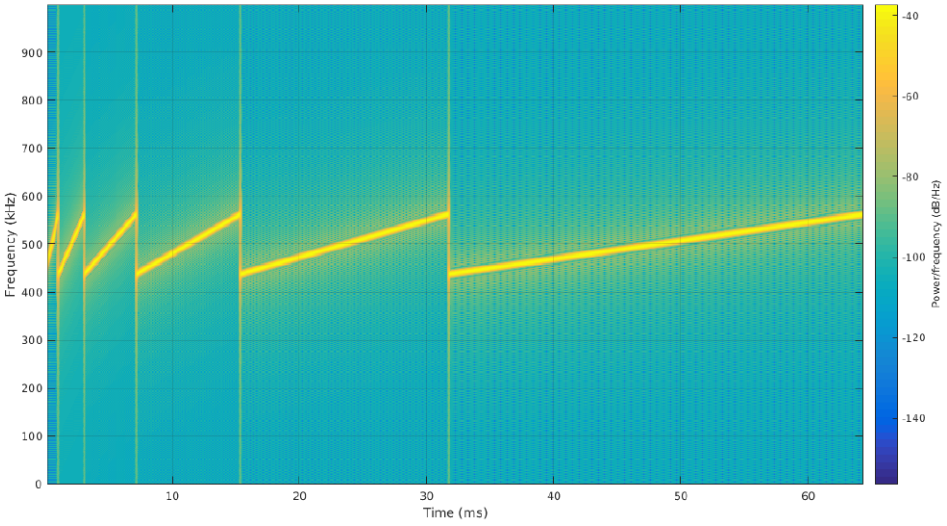


Figura 2.3: *Time on air* para diferentes *spreading factors*

O SF está também directamente relacionado com o tempo de transmissão. Num sistema de IoT com LoRaWAN a duração de transmissão é bastante importante, dado que está a operar na banda não-licenciada onde existe uma forte limitação no *duty-cycle*. Com o aumento do SF, o tempo de cada mensagem também aumenta levando ao incremento do tempo de transmissão. Com o tempo máximo de transmissão imposto pelo *duty-cycle* o aumento da duração de cada mensagem provoca uma redução do número de mensagens máximo que o sistema pode transmitir.

Para a abordagem à cobertura global apresentada pela rede LoRaWAN analisou-se uma infraestrutura de rede LoRaWAN especifica, neste caso a TTN visto que é de utilização gratuita.

Atualmente a rede LoRaWAN da TTN apresenta uma cobertura global bastante satisfatória inclusive em território nacional [3], tornando-se numa excelente solução para aplicações IoT, no entanto existem alguns pontos geográficos com falhas de cobertura. Nestas situações para estabelecer ligação a uma rede LoRaWAN é necessário instalar uma *gateway* LoRa. Esta característica pode parecer uma desvantagem mas, ao ser uma tecnologia de utilização completamente livre, também pode ser vista como uma vantagem, uma vez que qualquer pessoa pode instalar a sua própria *gateway* LoRa com um custo relativamente baixo (cerca de 100€ [4]) para uma instalação *indoor* ou um pouco

mais dispendioso (cerca de 300€ [5]) para uma instalação *outdoor* e assim conectar os seus dispositivos à rede global.

O LoRa/LoRaWAN não impõe um limite de mensagens *Uplink* e *Downlink*, mas ao estarmos a utilizar uma banda não licenciada é necessário cumprir o *duty-cycle* imposto que limita o recurso rádio utilizado.

Principais vantagens desta tecnologia:

- LoRaWAN opera no espectro electromagnético não-licenciado (livre de custos).
- Custo de aquisição de dispositivos terminais baixo (entre 3 a 5 €).
- LoRaWAN não é de uso exclusivo das operadoras móveis. Por exemplo a TTN é uma comunidade responsável pela gestão de diversas *gateways* LoRa.
- Utilização completamente livre o que possibilita a configuração e gestão da própria rede.
- Permite bidirecionalidade na comunicação devido à simetria na ligação *uplink* e *downlink*, tornando possível funcionalidade de controlo e comando de sistemas.
- Consumo energético reduzido.
- Dispositivos LoRa funcionam de forma satisfatória em movimento.
- Alcance entre 10 a 15 km em zonas rurais e 3 a 5 km em zonas urbanas [6].

Principais desvantagens desta tecnologia:

- Elevada complexidade do algoritmo implementado para que a transmissão de mensagens mais complexas seja feita com uma elevada eficiência (limitação do *duty-cycle*).

2.1.2 *Sigfox*

Outra tecnologia de comunicações de longo alcance é a *SigFox* [7], foi a primeira tecnologia LPWAN e utiliza uma banda não-licenciada, mas não é uma tecnologia livre uma vez que é necessário pagar uma subscrição para usufruir da rede.

Utiliza uma tecnologia de rádio patenteada com base no que é referido como *Ultra Narrow-Band* (UNB), como o próprio nome indica é um sistema com canais de banda estreita com uma modulação *Binary Phase-Shift Keying* (BPSK) numa banda de 100 Hz. Ao implementar uma modulação de banda estreita *Sigfox* torna a utilização da largura

de banda mais eficiente e promove níveis de ruído baixos o que leva a um baixo consumo energético, uma elevada sensibilidade do recetor e ao desenvolvimento de uma antena de baixo custo, tudo isto com uma taxa de transferência de apenas 100 bps [8].

A abordagem desta tecnologia ao utilizador final é difere da analisada na secção anterior, neste caso é o operador local que instala a infraestrutura de acesso radio, numa banda de frequência não licenciada. O resultado é uma rede global, gerida pelos operadores locais, composta por um conjunto de estações base com ligação direta às redes centrais de *SigFox*.

O princípio de funcionamento desta tecnologia é impor um ritmo binário muito baixo e um máximo de 140 mensagens *uplink* e 4 mensagens em *downlink* por dia (para cumprir os requisitos de *duty-cycle* existentes numa banda não-licenciada).

Inicialmente *SigFox* apenas suportava comunicação unidireccional, onde só a comunicação do nó para a *Gateway* (*uplink*) era permitida, com um tamanho máximo de 12 bytes por mensagem. Mais tarde foi implementada a comunicação bidireccional mas com uma capacidade muito reduzida, apenas 4 mensagens por dia em *Downlink* com um tamanho máximo de 8 bytes [8].

Principais vantagens desta tecnologia:

- Preço reduzido dos dispositivos terminais (inferior a 2€);
- Baixo consumo energético;
- Bom funcionamento para dispositivos transmitam informação de forma esporádica.
- Alcance entre 30 a 50 km em zonas rurais e 3 a 10 km em zonas urbanas [6].
- Cobertura europeia, subscrição é global.

Principais desvantagens desta tecnologia:

- Comunicação bidireccional pouco satisfatória, dada a reduzida capacidade na ligação *downlink*.
- Utilização da tecnologia implica o pagamento de uma subscrição a uma operadora local.
- O utilizador está dependente da operadora para solucionar ou mitigar eventuais falhas de cobertura do serviço, uma vez que a manutenção e gestão da infraestrutura rádio é responsabilidade da operadora.

2.1.3 LTE-M

Apresentadas as duas tecnologias de operação em banda não licenciada, é agora necessário abordar as tecnologias baseadas na rede de operador, ou seja, implementadas na banda licenciada. Estas soluções assentes na rede de operador, retiram da responsabilidade do utilizador a criação e gestão de uma rede IoT para que este apenas adquira o serviço de transmissão de informação.

Ao longo do tempo as tecnologias celulares tradicionais foram-se adaptando para responder às necessidades IoT (baixo consumo energético e grande cobertura).

Surge então uma versão do *Long-Term Evolution* (LTE), o LTE-M para responder às exigências de um sistema IoT. Um dos aspectos otimizado, face à versão de LTE normal, foi a possibilidade de um terminal permanecer longos períodos sem comunicar, poupando assim a bateria do dispositivo.

Esta tecnologia já está disponível em alguns países europeus mas ainda não está implementada em Portugal.

Principais vantagens desta tecnologia:

- Sem limitações impostas pelo *duty-cycle*;
- Ritmo de transmissão elevado (até 1 Mbps) [9];
- Esforço de implementação (por parte dos operadores) reduzido uma vez que utiliza grande parte da infraestrutura existente do LTE;

Principais desvantagens desta tecnologia:

- Necessidade de subscrição (serviço fornecido por uma operadora);
- Sem implementação em Portugal;

2.1.4 NB-IoT

NB-IoT é uma tecnologia IoT de banda estreita especificada na versão 13 da *3rd Generation Partnership Project* (3GPP) em Junho de 2016. NB-IoT pode coexistir com *Global System for Mobile Communications* (GSM) e LTE operando nas bandas de frequência licenciadas (por exemplo: 700MHz, 800 MHz, e 900 MHz). Os pacotes em *uplink* podem ir até 125 bytes e até 85 bytes para *downlink* [10]. Ocupa uma largura de banda de 200 KHz, o que corresponde a uma sub-portadora em GSM e LTE [11] que as operadoras têm reservada especificamente para sistemas IoT.

O protocolo de comunicação da *NB-IoT* é baseado no protocolo presente no LTE com modulação BPSK. A estratégia utilizada foi reduzir ao mínimo as funcionalidades do protocolo LTE e e melhorar aquelas necessárias para aplicações IoT. Por exemplo, o sistema de *backend* do LTE é utilizado para fazer o *broadcast* de informação útil para todos os dispositivos terminais dentro da célula. Uma vez que este sistema de*broadcast* consome recursos rádio e bateria de cada um dos dispositivos terminais, é reduzida ao máximo tanto em duração como em ocorrência. Foi otimizado para enviar pequenas mensagens e para evitar funcionalidades desnecessárias para a implementação IoT. Com isto é possível reduzir de forma substancial o consumo energético dos vários dispositivos terminais[1].

Já se encontra disponível em Portugal apesar da cobertura ainda apresentar lacunas em alguns locais (alcance até 10 km) [12], aspecto que pode mudar com a melhoria constante da oferta por parte dos operadores.

Principais vantagens desta tecnologia:

- Sem limitações impostas pelo *duty-cycle*;
- Baixo consumo de energia;
- Elevada fiabilidade;
- Suporta posicionamento geográfico sem a necessidade de utilização de GPS.

Principais desvantagens desta tecnologia:

- Necessidade de subscrição (serviço fornecido por uma operadora);
- Cobertura ainda muito limitada em território nacional;

2.1.5 EC-GSM

Tecnologia baseada em 2G e por isso é sempre uma tecnologia a ter em conta dada a grande cobertura em território nacional, em alguns pontos sendo a única existente. Mas infelizmente ainda não está disponível em Portugal.

Principais vantagens desta tecnologia:

- Sem limitações impostas pelo *duty-cycle*;
- Esforço de implementação (por parte dos operadores) reduzido uma vez que utiliza grande parte da infraestrutura existente do 2G, sendo apenas necessário configurações de *software*;

Principais desvantagens desta tecnologia:

- Necessidade de subscrição (serviço fornecido por uma operadora);
- Sem implementação em Portugal;

As principais vantagens da utilização de tecnologias celulares para sistemas IoT é a inexistência de qualquer limitação de *duty-cycle* e a abstração de tarefas relacionadas com a gestão da rede porque o utilizador está a pagar um serviço a um operador que por sua vez disponibiliza um recurso rádio é responsável pela gestão do recurso.

2.1.6 Comparação

De forma a enumerar os aspectos a reter nesta secção e a justificar a opção implementada apresenta-se o seguinte quadro comparativo, para as tecnologias LPWAN abordadas e de possível implementação (excluindo EC-GSM e LTE-M)

	LoRaWAN	***Sigfox***	***NB-IoT***
Modulação	*CSS*	*BPSK*	*QPSK*
Tecnologia Rádio	Espalhamento Espectral	UNB	*OFDM*
Alcance	< 20 km	> 40 km	< 10 km
Espectro	Banda não licenciada (863–870 MHz) Europa	Banda não licenciada (868 Mhz) Europa	Banda LTE
Largura de Banda	125 a 250 kHz	100 Hz	200 kHz
Ritmo Binário	até 50 kbps	até 100 bps	até 200 kbps
Bidirecional	Sim	Sim (com limitações)	Sim
Mensagens por dia	Restrição de *duty-cycle*	até 140(*UL*), 4(*DL*)	Sem limitação
Tamanho da Mensagem	até 243 bytes	até 12 bytes(*UL*), 8 bytes (*DL*)	até 125 bytes(*UL*), 85 bytes (*DL*)
Autenticação/ Encriptação	Não Suportado	Sim	Sim

Tabela 2.1: Quadro comparativo das tecnologias *Sigfox*, *LoRaWAN*, *NB-IoT*

Outro aspeto a ter em conta para justificar a opção tomada é a comparação dos vários custos inerentes a cada tecnologia (dispositivos, instalação e espectro).

	LoRa/LoRaWAN	***Sigfox***	***NB-IoT***
Espectro	Sem custo	Sem custo	Banda Licenciada
Implementação	De 100€ - 300€	A cargo da operadora	A cargo da operadora
Dispositivo Terminal	3 - 5€	< 2€	> 20€

Tabela 2.2: Comparação de custos das tecnologias *Sigfox*, *LoRaWAN*, *NB-IoT*

Os principais pontos a reter:

- Qualidade de serviço e capacidade disponibilizada

Neste aspecto a tecnologia NB-IoT leva, claramente, vantagem sobre as outras duas fruto essencialmente a utilização de uma largura de banda licenciada e de um protocolo baseado em LTE [1]. O LoRa e *Sigfox* vêem as suas capacidades bastante reduzidas para conseguir cumprir com as limitações impostas pelo *duty-cycle*, fator que se acentua no caso do *Sigfox*.

- Cobertura e alcance

Esta é a característica mais interessante na utilização das tecnologias LoRa e *Sigfox*, especialmente da última onde uma única estação base consegue alcançar raios de cobertura superiores a 40 km, contrastando com o LoRa que apresenta um alcance inferior a 20 km. NB-IoT é a tecnologia com a capacidade de cobertura e alcance mais reduzida (inferior a 10 Km), tendo a particularidade de não ser adequado para regiões que não beneficiam da cobertura de LTE uma vez que a implementação de NB-IoT está limitada à cobertura oferecida pelas estações base LTE[1].

- Custos

Relativamente aos custos apenas a LoRaWAN é de utilização gratuita bastando para isso a aquisição de um terminal LoRa/LoRaWAN. Tanto o *Sigfox* como o *NB-IoT* exigem uma subscrição para ter acesso à tecnologia .

- Conclusões

Dadas as características e particularidades expostas e tendo em conta o objectivo do projecto em causa a escolha recai sobre a LoRa/LoRaWAN essencialmente pela independência das operadoras locais para a distribuição rádio e gestão da infraestrutura IoT, pela possibilidade de utilização de uma comunicação bidireccional, por não exigir uma subscrição pela utilização dos recursos rádio.

2.2 Infraestruturas de Rede *IoT*

Uma vez definida o LoRaWAN como tecnologia de ligação e comunicação entre a aplicação cliente e a rede de dispositivos é também importante definir a infraestrutura que dará suporte à rede LoRaWAN. Esta infraestrutura permite a conectividade, gestão e monitorização de dispositivos, *gateways* e aplicações clientes. O seu principal objectivo é garantir a segurança, escalabilidade e fiabilidade do encaminhamento dos pacotes para toda a rede. Este tipo de infraestruturas são geralmente baseadas em soluções de *cloud*.

Nesta secção serão abordadas três opções: TTN, *LORIOT* e *Helium* e qual delas se adapta melhor ao projecto implementado.

2.2.1 *LORIOT*

É uma infraestrutura de rede construída com o objectivo de oferecer um serviço de gestão e controlo de uma rede LoRaWAN através do pagamento de uma subscrição. Apresenta um forte apoio ao cliente e sem qualquer limitação no número de dispositivos registados. Existe a possibilidade de usufruir de forma gratuita mas com muitas limitações, apenas com a possibilidade de utilização de 10 dispositivos IoT e sem qualquer suporte técnico.

Principais vantagens:

- Estrutura de apoio ao cliente bastante sólida;
- Vasto leque de funcionalidades para a gestão e controlo dos dispositivos da rede IoT implementada;
- Interface em *web browser* para gestão e processamento de dados

Principais desvantagens:

- Para usufruir totalmente das varias funcionalidades oferecidas pela *LORIOT* é necessário pagar uma subscrição;

- Versão gratuita muito limitada, quando comparada com as versões pagas, ao nível dos dispositivos IoT disponíveis, aplicações e suporte técnico.

2.2.2 *Helium*

Base de funcionamento semelhante à infraestrutura apresentada anteriormente, ou seja oferece um serviço de gestão de rede com custos para o utilizador. O método utilizado assenta no pagamento de um determinado valor por cada mensagem enviada. Tem como principal mercado os Estados Unidos da América onde apresenta uma forte cobertura, apesar de actualmente existirem algumas *gateways Helium* em Portugal na zona de Lisboa.

Principais vantagens:

- Estrutura de apoio ao cliente satisfatória;
- Interface em *web browser* bastante intuitiva e acessível para gestão e controlo dos vários dispositivos IoT;

Principal desvantagem:

- Apesar de não ser necessário pagamento uma subscrição para utilizar a plataforma, cada mensagem trocada entre os vários dispositivos IoT acarreta um custo para o utilizador;

2.2.3 *The Things Network*

Ao contrario das soluções anteriores, a TTN oferece uma infraestrutura LoRaWAN sem necessidade do pagamento de uma subscrição, sendo possível fazer o controlo e gestão de dispositivos IoT de forma completamente gratuita. Sem limite de dispositivos registados na rede e com um bom suporte técnico alicerçado na comunidade de utilizadores.

Independentemente da escolha é importante salientar que num ambiente onde existam diversos dispositivos IoT mesmo que configurados em diferentes infraestruturas de rede, quando um nó de LoRaWAN faz *broadcast* de uma mensagem todos os *gateways* na área de cobertura vão receber o pacote LoRa enviado, a *gateway* faz o reencaminhamento do pacote para a rede que estiver configurada (uma *gateway TTN* faz o reencaminhamento de pacotes para a rede TTN e uma *gateway loriot* faz o reencaminhamento de pacotes para a rede *LORIOT*). Esta situação não apresenta qualquer

problema porque os pacotes LoRaWAN são encriptados, por isso quando uma rede *LORIOT* recebe um pacote de um nó configurado para TTN não o vai conseguir desencriptar e vice-versa.

Principais vantagens:

- Possibilita a gestão e controlo de uma rede IoT de forma completamente gratuita sem qualquer tipo de limitação;
- Interface em *web browser* bastante intuitiva e acessível para gestão e controlo dos vários dispositivos IoT;
- Compatibilidade com ferramentas de programação permitindo ao utilizador adaptar o processamento de dados de acordo com as suas necessidades;
- Cobertura satisfatória em território nacional (especialmente na zona litoral).

Principal desvantagem:

- O facto de ser uma plataforma totalmente gratuita o suporte técnico nem sempre está garantido

2.2.4 Comparação

De forma a enumerar os aspectos a reter nesta secção e a justificar a opção implementada apresenta-se o seguinte quadro comparativo.

	TTN	***LORIOT***	***Helium***
Custo	Gratuito	Subscrição mensal	Pagamento por mensagem
Nº Dispositivos Terminais	Sem limite	10 (gratuitos)	Sem limite
Nº *Gateways*	Sem limite	1 (gratuito)	Sem limite
Suporte Técnico	Suportado pelos utilizadores	Dependente do pacote escolhido	Suportado pelos utilizadores
Cobertura do Serviço	Satisfatório	Dependente do pacote escolhido	Satisfatório

Tabela 2.3: Quadro comparativo das infraestruras de rede para LoRaWAN

Tendo em conta os objectivos do projecto a infraestrutura de gestão da rede LoRaWAN escolhida foi a TTN. É uma rede completamente gratuita, possibilita bastantes soluções de monitorização e interacção com os dispositivos IoT da rede implementada e apresenta uma cobertura interessante em Portugal, especialmente em Lisboa.

2.3 Plataforma Aplicacional

No âmbito do controlo e gestão de uma rede de dispositivos uma etapa importante é o desenvolvimento de uma aplicação, sendo esta o interface visual do utilizador com a rede de dispositivos. Tem como principal função o processamento de dados inseridos pelo utilizador de forma a possibilitar a interacção com a infraestrutura LoRaWAN que por sua vez transmite a mensagem à rede de dispositivos.

Nesta secção serão analisadas duas ferramentas/plataformas possíveis para o desenvolvimento de uma aplicação. Tendo como principal requisito a facilidade de desenvolvimento e implementação optou-se por ferramentas/plataformas que apresentem uma forte componente de "programação visual", ou seja, programação baseada em blocos e fluxos. As plataformas em discussão serão: *Node-RED* e *ThingsBoard*.

2.3.1 *Node-RED*

Node-RED é uma aplicação baseada em *Node.js*, um ambiente de execução projetado para desenvolvimento de aplicações em *JavaScript*. Isto significa que ao utilizar o *Node.js* é possível executar uma aplicação *JavaScript* na maquina local sem a dependência de um *browser* para a executar.

O *Node-RED* fornece uma *Graphical User Interface* (GUI) onde os utilizadores arrastam e largam blocos que representam componentes de um sistema maior. Entre estes componentes podem ser colocados outros blocos para representar funções de *software* para o processamento da informação entre os componentes do sistema.

Cada um dos blocos que se observam na figura 2.4, são uma representação visual de um bloco de código *JavaScript* concebido para realizar uma tarefa específica.

Principais vantagens:

- Interface de programação visual bastante intuitiva, facilitando a integração de sistemas mais complexos;
- *Node-RED* disponibiliza informação detalhada para a apoiar a programação no seu ambiente;

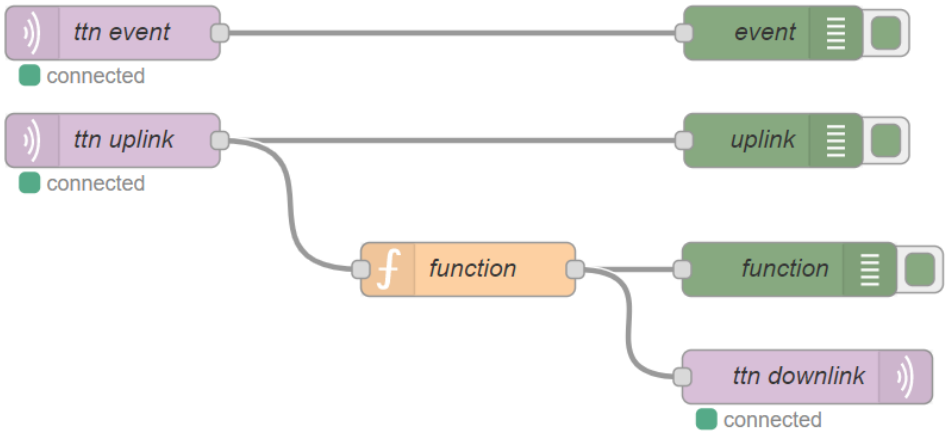


Figura 2.4: Exemplo da plataforma *Node-RED*

- Integração com diversas plataformas, incluindo o TTN, de forma completamente gratuita;
- Sem custos para o utilizador;

Principal desvantagem:

- Para executar ou desenvolver uma aplicação em *Node-RED* é necessário instalar uma biblioteca ou módulo *Node.js*. O que pode aumentar, de forma ligeira, a complexidade da integração de uma solução em *Node-RED*.

2.3.2 *ThingsBoard*

ThingsBoard é uma plataforma de desenvolvimento, gestão e dimensionamento de projectos IoT. Como é possível observar na figura 2.5 o método de programação adotado é bastante semelhante ao da ferramenta anterior, ou seja, é feito através da manipulação e interligação de diversos blocos de código.

A *ThingsBoard* permite a integração, de forma direta, com diversas plataformas externas entre as quais a TTN mas esta funcionalidade apenas está disponível na versão profissional.

Principais vantagens:

- Interface de programação visual bastante intuitiva, facilitando a integração de sistemas mais complexos;
- Disponibiliza uma interface *web* para o desenvolvimento ou gestão da aplicação;

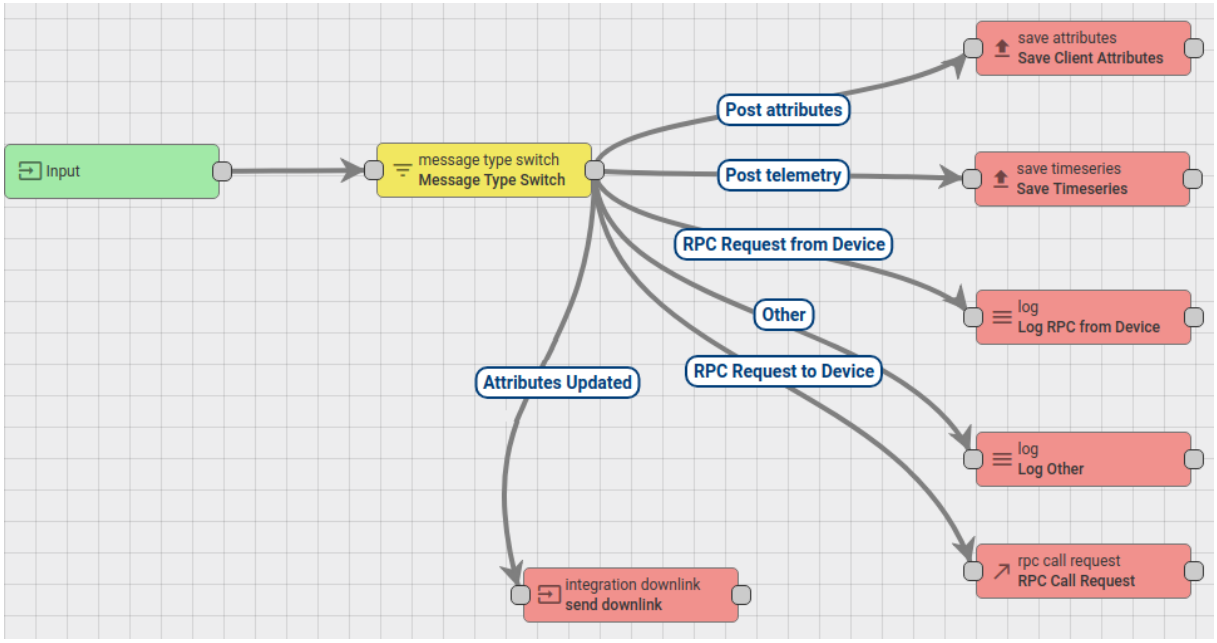


Figura 2.5: Exemplo da plataforma *ThingsBoard*

Principal desvantagem:

- Versão gratuita bastante limitada, nomeadamente na integração com plataformas externas. Estando apenas disponível na versão profissional.

2.3.3 Comparação

Ambas as plataformas aqui apresentadas permitem a monitorização e controlo de dispositivos IoT através do processamento de dados enviados e recolhidos da rede de dispositivos IoT. A sua configuração é também bastante semelhante, é feita através da interligação de vários blocos/componentes que na sua totalidade formam um sistema global.

A base de funcionamento deste tipo de sistemas consiste na recolha inicial de uma mensagem/evento proveniente da rede de dispositivos ou da interacção do utilizador com a própria aplicação, a mensagem percorre os vários nós/blocos e vai sendo processada e enviada para o bloco seguinte consoante a regra/função implementada no bloco. A vantagem deste tipo de plataformas é a facilidade de configuração e implementação de um sistema de controlo IoT, fruto essencialmente da forma intuitiva e visual com que se interliga os diversos blocos, reduzindo de forma substancial a necessidade de implementação de código escrito.

Para o desenvolvimento deste projecto e tendo em conta a infraestrutura de rede utilizada (TTN) a plataforma escolhida para o desenvolvimento de uma aplicação de interacção do utilizador com a rede IoT foi o *Node-RED*.

Esta plataforma apresenta algumas vantagens quando comparada com a *Thingsboard*:

- A plataforma *Node-RED* disponibiliza todas as suas funcionalidade de forma completamente gratuita;
- A plataforma *Node-RED* permite a integração simples e direta com a TTN;
- Interface de configuração e programação dos diferentes nós mais intuitiva e menos complexa.

Estes aspectos tornam a utilização do *Node-Red*, no âmbito deste projecto, mais vantajoso que outra aplicação deste tipo.

2.4 Micro-controladores

Uma área de elevada importância para a implementação deste projecto é a escolha do micro-controlador que incorpora os vários sensores/dispositivos do sistema. Antes do micro-controlador ser escolhido é necessário definir o protocolo de comunicação *wireless* entre dispositivos, porque o micro-controlador escolhido depende do tipo de protocolo de comunicação utilizado.

Tendo em conta o enquadramento e requisitos deste projecto:

- O consumo energético não é um factor limitativo;
- O alcance de sinal é um aspecto a ter em conta dada a possível distribuição física dos dispositivos.

Optou-se, então, por implementar o sistema de micro-controladores assente numa rede *Wi-Fi*, visto que apresenta um alcance de sinal superior quando comparado com outros protocolos mesmo que para isso exija um maior gasto energético (como por exemplo o *Bluetooth Low Energy* (BLE) que consome menos energia mas por outro lado apresenta uma cobertura de sinal inferior). Seleccionou-se então, para análise, um conjunto de micro-controladores capazes de integrar e satisfazer uma rede *Wi-Fi*. Os micro-controladores escolhidos foram: ESP32 e ATmega4809.

2.4.1 ATmega4809

Este micro-controlador equipa uma das mais recentes placas *Arduino*, o *Arduino Uno WiFi Rev2* que tem como principal característica a possibilidade de estabelecer ligações *Wi-Fi* sem necessidade de um módulo externo. O ATmega4809 possui as seguintes características[13]:

- Memória Flash: 48KB;
- Memória SRAM: 6KB;
- Memória EEPROM: 256 Bytes;

E visto que o ATmega4809 não seria utilizado de forma isolada, é também importante nomear as principais características inerentes à placa *Arduino Uno WiFi Rev2*[13]:

- Consumo energético: 50mA - 150mA;
- Pins Digitais I/O: 14;
- Frequência do relógio: 16MHz;
- Tensão de operação: 5v.

Uma das principais vantagens na escolha de uma placa *Arduino* é a fácil e intuitiva utilização do ambiente de programação disponibilizado e a existência de bastante suporte tanto a nível de exemplos disponíveis como de diversos fóruns de discussão. Como desvantagem está a pouca memória disponibilizada e o elevado custo de aquisição (aproximadamente 40€[13]).

Principais vantagens:

- Facilidade de programação e implementação num projecto;
- Fóruns de discussão técnica muito ativos;

Principais desvantagens:

- Elevado custo de aquisição;
- Poucos recursos de memória;

2.4.2 ESP32

O ESP32 é uma evolução do ESP8266, tendo sido concebido pela empresa chinesa *Espressif*. É um dispositivo constituído por um micro-processador e uma interface *Wi-Fi* e *Bluetooth* tudo integrado na mesma placa. É por isso de fácil percepção que o objectivo principal da criação deste dispositivo é potenciar a conectividade *wireless* entre os diversos elementos de um sistema [14]. Actualmente existem diversos protocolos de comunicação *wireless*, desenvolvidos pela *Espressif*, que suportam e facilitam a construção/inicialização e gestão de uma rede *wireless*.

O ESP32 possui as seguintes características[15]:

- Endereçamento: até 4Gb (32 bit)
- Memória Flash: até 16Mb;
- Memória SRAM: 520 KB;
- Memória ROM: 448 KB;
- Frequência do relógio: 80MHz (regime mínimo) até 160MHz (máximo);
- Pins digitais I/O: 36;
- Consumo energético: 20mA a 240mA;
- Tensão de operação: 3.3V.

O ESP32 é um sistema *dual-core* com dois CPUs *Xtensa LX6* com arquitectura *Harvard*. O mapeamento de endereços destes dois CPUs é simétrico, ou seja, utilizam os mesmos endereços para aceder à mesmo recurso de memória[15]. Esta característica permite ao ESP32 atingir níveis de performance superiores quando comparado com o seu antecessor (ESP8266)[14].

Outro aspecto importante deste dispositivo é também a facilidade programação, uma vez que a *Espressif* fez questão de tornar possível a programação destes dispositivos através da mesma plataforma utilizada pelo *Arduino*, sendo por isso possível usar grande parte parte das bibliotecas existentes para as placas *Arduino*.

Principais vantagens:

- Baixo custo de aquisição;
- ESP32 foi desenvolvido tendo como principal objectivo a conectividade *wireless*;

- Capacidade de processamento elevada;
- Recursos de memória bastante satisfatórios;
- Frequência de trabalho elevada;
- Informação técnica fornecida bastante detalhada;

Principal desvantagem:

- Fóruns de discussão técnica com menos variedade de tópicos abordados (quando comparado com o micro-controlador abordado anteriormente), o que pode dificultar o esclarecimento de algumas duvidas que possam surgir durante a implementação do projeto.

2.4.3 Comparação

De forma a enumerar os aspectos a reter nesta secção e a justificar a opção implementada apresenta-se o seguinte quadro comparativo, assumindo que o micro-controlador ATmega4809 será utilizado com a placa *Arduino Uno WiFi Rev2*.

	ESP32	**ATmega4809**
Custo	7 - 10€	40€
Memória Flash	4 a 16Mb	48 KB
Memória RAM	520 KB	6 KB
Memória ROM	448 KB	256 Bytes
Pins Digitais I/O	36	14
Tensão de operação	3.3V	5V
Consumo Energético	20mA a 240mA	50mA a 150mA

Tabela 2.4: Quadro comparativo dos micro-controladores

Tendo em conta a análise feita aos dispositivos mencionados nesta secção optou-se pela escolha da placa ESP32, essencialmente pela elevada capacidade de processamento e memória oferecida por este micro-controlador. Este aspecto tem particular importância no contexto do projecto desenvolvido dada a elevada exigência de memória do mesmo (fruto, por exemplo,da eventual necessidade da implementação de uma base de dados nos dispositivos ou outro tipo). Outro factor levado em conta foi o custo de aquisição, placas ESP32 apresentam um custo de aquisição substancialmente inferior ao das

placas *Arduino*, este factor torna-se ainda mais relevante tendo em conta que o sistema não tem um numero definido de membros. Relativamente à programação dos dispositivos, apesar das placas *Arduino* oferecerem uma programação mais acessível (fruto do maior suporte e bibliotecas existentes), utilizando a mesma plataforma (*Arduino IDE*) a curva de aprendizagem para a programação da placa *ESP32* é pequena, não sendo por isso um factor determinante para a escolha de *Hardware*. Este micro-controlador está disponível em diversos formatos, o selecionado para este projeto foi o *ESP32-DevKitC* que permite a montagem em *bread board* o que facilita e agiliza a realização dos testes necessários.

2.5 Protocolo de Comunicação de curta distância

No capítulo 2.4 definiu-se o micro-controlador que seria utilizado neste projecto (o ESP32). Uma vez que o micro-controlador escolhido foi o *ESP32* tirou-se partido de dois protocolos de comunicação desenvolvidos pela *Espressif*: o *ESP-Mesh* e o *ESP-BLE-Mesh*. É em torno destes dois protocolos que incidirá esta secção.

2.5.1 *ESP-Mesh*

ESP-Mesh é um protocolo de rede construído sobre o protocolo de *Wi-Fi*. Este protocolo permite que vários dispositivos (distribuídos por uma vasta área física) estejam interligados sob uma *Wireless LAN* (WLAN). O protocolo *ESP-Mesh* possibilita que a rede, através dos vários dispositivos que a constituem, seja construída e mantida de forma autónoma[16].

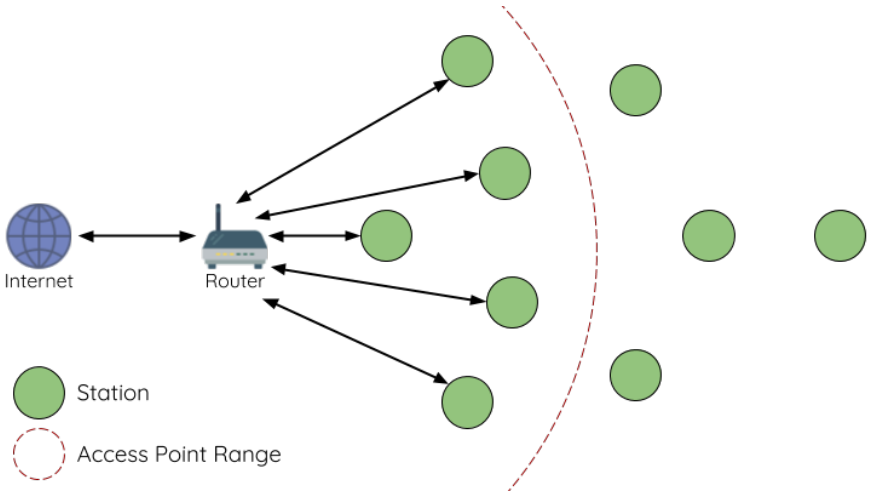


Figura 2.6: Rede tradicional *Wi-Fi*[16]

A estrutura típica de uma rede *Wi-Fi* (figura 2.6) é *point-to-multipoint* onde um único nó central, conhecido como *Access Point* (AP), está diretamente ligado a todos os outros nós, também conhecidos como estações. O AP é responsável por gerir e reencaminhar as transmissões entre estações. Alguns APs também reencaminham transmissões de/para uma rede externa através de um *router*.

Uma rede tradicional de *Wi-Fi* apresenta a grande desvantagem de uma área de cobertura limitada, uma vez que todas as estações têm que se ligar diretamente ao AP. Outra desvantagem é o facto do número de estações da rede estar limitado pela capacidade do AP[16].

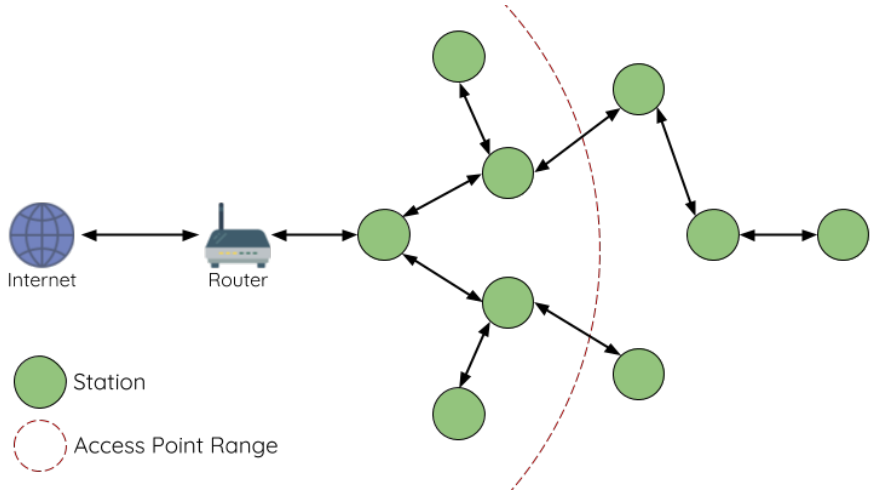


Figura 2.7: Rede *ESP-Mesh*[16]

A grande diferença de uma rede baseada no protocolo *ESP-Mesh* (figura 2.7) para uma rede *Wi-Fi* tradicional reside na necessidade de cada nó estar ligado ao nó central, uma vez que é permitido a cada nó conectar-se ao nó vizinho. Os vários nós da rede são responsáveis por retransmitir as diversas comunicações uns dos outros, o que permite alcançar áreas de cobertura muito superiores, tendo em conta que para pertencer à rede um dispositivo tem apenas de estar ao alcance do nó mais próximo. Outra importante diferença é a capacidade máxima não ser limitada pelo nó central[16].

ESP-Mesh pode ser visto como um protocolo de rede que combina várias redes *Wi-Fi* individuais numa única WLAN. Em *Wi-Fi* as estações estão limitadas a uma única ligação ao AP (*Uplink*), enquanto um AP pode estar conectado a várias estações em simultâneo (*downlink*). No entanto o *ESP-Mesh* permite que um nó seja, simultaneamente, uma estação e um AP. Com isto, um nó numa rede *ESP-Mesh* pode ter múltiplas ligações em *downlink* utilizando a uma interface *softAP* e, ao mesmo tempo, ter uma ligação *uplink* utilizando a sua interface de "estação".

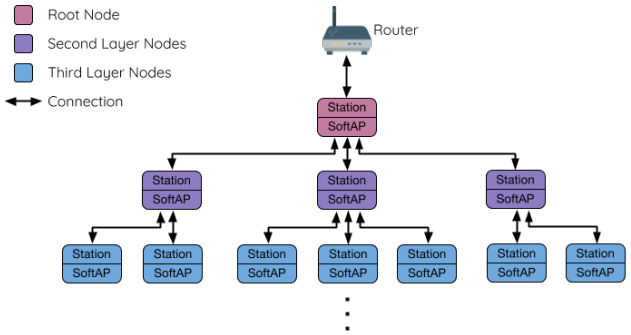


Figura 2.8: *ESP-Mesh* Tree Topology[16]

ESP-Mesh é uma rede *multi-hop*, ou seja, cada nó tem a capacidade de transmitir os pacotes para outros nós na rede através de um ou mais "saltos". Assim, os nós numa rede *ESP-Mesh* não só transmitem apenas os seus pacotes, como também servem de caminho para outros nós. Esta característica permite, desde que exista um caminho entre dois nós na camada física, que qualquer par de nós numa rede *ESP-Mesh* possa comunicar.

Cada nó do ESP-Mesh capaz de formar conexões *downstream* (ou seja, com uma interface softAP) transmitirá periodicamente *Beacon Frames* Wi-Fi. Um nó utiliza *Beacon Frames* para permitir que outros nós detetem a sua presença e saibam do seu estado. Os nós *Idle* ouvirão os *frames beacon* para gerar uma lista de potenciais nós para se conectar, será com um desses nós que formará uma conexão *upstream*[16].

A força de sinal de uma possível ligação *upstream* é representada pela *Received Signal Strength Indication* (RSSI) da *beacon frame* do possível nó que poderá receber a ligação. Com o objetivo de prevenir a formação de ligações *upstream* demasiado fracas, foi implementado um mecanismo de *threshold* para as *beacon frames*. Se um nó detetar uma *beacon frame* com uma RSSI abaixo do *threshold* pré-configurado, a ligação *upstream* formada não será considerada.

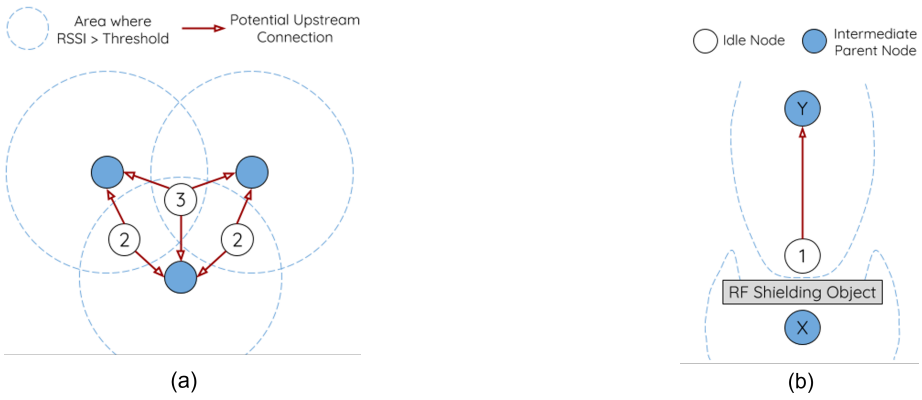


Figura 2.9: Efeitos do mecanismo de *threshold* da *RSSI*[16]

Na Figura 2.9 podemos observar os efeitos provocados pelo mecanismo de *threshold* da RSSI.

- Figura 2.9 (a) : Nesta figura é possível observar-se como o *threshold RSSI* afeta o número de possíveis ligações apresentadas ao nó *idle*.

- Figura 2.9 (b) : Este cenário demonstra como a presença de um obstáculo pode reduzir de forma significativa o RSSI de um possível nó pai. Devido a este obstáculo, a área onde o RSSI está acima do *threshold* também sofre uma redução significativa. No caso da Figura B, observamos que o nó *idle* forma uma ligação *upstream* com o nó Y pois apresenta um melhor RSSI mesmo que fisicamente mais distante.

As grandes vantagens a destacar com a utilização deste protocolo são, a gestão autónoma da rede e a interacção dinâmica dos vários elementos que a constituem (*multi-hop*), que se traduz num aumento de eficiência e alcance da transmissão de dados face a uma implementação tradicional *Wi-Fi*.

Principais vantagens com a utilização do protocolo *ESP-Mesh*:

- Gestão autónoma da rede;
- Suporta mecanismos de *multi-hop*, que se traduz num aumento de eficiência e alcance da transmissão de dados face a uma implementação tradicional *Wi-Fi*;
- Rede com elevada fiabilidade. A redundância de ligações permite que a probabilidade de quebra de ligação entre dois dispositivos seja reduzida devido à existência de vários caminhos para o mesmo destinatário.

Principais desvantagens da utilização do *ESP-Mesh*:

- Complexidade de configuração de cada nó da rede *Mesh*;
- Consumo energético. O facto de cada nó ser ao mesmo tempo um AP e um *endpoint* pode resultar num aumento do consumo de energia;

2.5.2 *ESP-BLE-Mesh*

Semelhante ao clássico *Bluetooth*, o BLE opera na banda não licenciada dos 2.4 Ghz. No entanto para reduzir o custo e o consumo energético, o BLE utiliza uma modulação de frequência binária com 1 Mbit/s. Ao contrário do *Bluetooth*, que utiliza 79 canais

com 1 Mhz de largura de banda, o BLE utiliza 40 canais com 2 Mhz de largura de banda. Dos 40 canais, 3 são para a descoberta de novos dispositivos e estabelecer ligação (*advertising channels*), os restantes 37 são utilizados para a transferência de dados entre dispositivos [17]. O BLE é capaz de reconhecer novos dispositivos apenas com a transmissão de pacotes de *advertising* de 1 em 1 segundo, esta funcionalidade reduz de forma significativa o consumo energético [18].

Em 2017, o *Bluetooth Special Interest Group* (SIG) anunciou a desenvolvimento de um protocolo de rede, sobre o protocolo de comunicação BLE, o *Bluetooth Mesh* que suporta a implementação de dispositivos *Bluetooth* numa rede *Mesh* [19].

O *Bluetooth Mesh* não é uma tecnologia *wireless*, mas sim uma tecnologia de rede. Esta tecnologia depende do BLE - um protocolo de comunicação *wireless*. O *ESP-BLE-MESH*, desenvolvido pela *Espressif* assente no *Zephyr Bluetooth Mesh*, suporta a gestão de dispositivos *BLE* e o controlo dos nós da rede *Mesh*. [20].

Principais vantagens na utilização do *ESP-BLE-Mesh*:

- Baixo consumo energético;
- Suporta mecanismos de *multi-hop*, que se traduz num aumento de eficiência alcance da transmissão de dados face a uma implementação tradicional de *Bluetooth*;
- Rede com elevada fiabilidade.

Principais desvantagens na utilização do *ESP-BLE-Mesh*:

- No contexto deste projeto, o facto do *BLE* ser um protocolo para comunicação a curta distância pode ser visto como uma desvantagem;
- Ao utilizar o *ESP-BLE-Mesh* implementação de dispositivos *BLE* na rede *Mesh* não é automática. O utilizador necessita de, através de uma aplicação, selecionar individualmente cada dispositivo para formar e integrar a rede *Mesh* [20].

2.5.3 Comparação

De forma a enumerar os aspectos a reter nesta secção e a justificar a opção implementada apresenta-se o seguinte quadro comparativo:

	ESP-Mesh	***ESP-BLE-Mesh***
Alcance	até 100m (*Wi-Fi*)	de 10m - 20m (*Bluetooth*)
Consumo Energético	Superior ao *BLE*	Baixo
Implementação	Automática	Manual

Tabela 2.5: Quadro comparativo entre protocolos

Dadas as características analisadas o protocolo de comunicação de curta distância selecionado para o desenvolvimento deste projeto foi o *ESP-Mesh*, uma vez que apresenta vantagens importantes face ao *ESP-BLE-Mesh*: ao estar assente sobre o protocolo *Wi-fi* apresenta um alcance superior e dados os requisitos do projeto (a distancia entre dispositivos pode ser na casa das dezenas de metros) este fator acaba por ser determinante na escolha do protocolo a utilizar, outra característica a ter em conta é a implementação da rede *Mesh* e mais uma vez o *ESP-Mesh* tem vantagem ao apresentar uma gestão totalmente autónoma ao contrario do *ESP-BLE-Mesh* que necessita de intervenção do utilizador.

2.6 Trabalho Relacionado

Atualmente existe disponível alguma oferta de softwares/plataformas de controlo e monitorização, especialmente desenvolvidos para a implementação em meio artístico com o objetivo de facilitar a comunicação entre os vários elementos físicos e digitais.

2.6.1 Isadora

Isadora é um software criado por *Mark Coniglio* concebido para permitir a manipulação interactiva e em tempo real de uma vasta gama de conteúdos multimédia, bem como outros dados produzidos pelos sensores.

O *Isadora* oferece ao utilizador um ambiente de programação gráfica para o controlo dos vários elementos que compõem o sistema. Baseia-se na interligação de vários módulos, cada um dos quais com uma função específica no sistema. Estes módulos podem ser configurados para processar informação proveniente de sensores ou de uma interface *Musical Instrument Digital Interface* (MIDI) e, desta forma, influenciar o fluxo de ações do sistema. É uma ferramenta de elevada complexidade com requisitos de funcionamento exigentes (16GB de RAM e processador Intel i7 ou equivalente[21]). Outro aspeto a ter em consideração é o custo de subscrição, cerca de 150€ anuais ou 17,5€ mensais com novas atualizações a serem pagas em separado([22]).

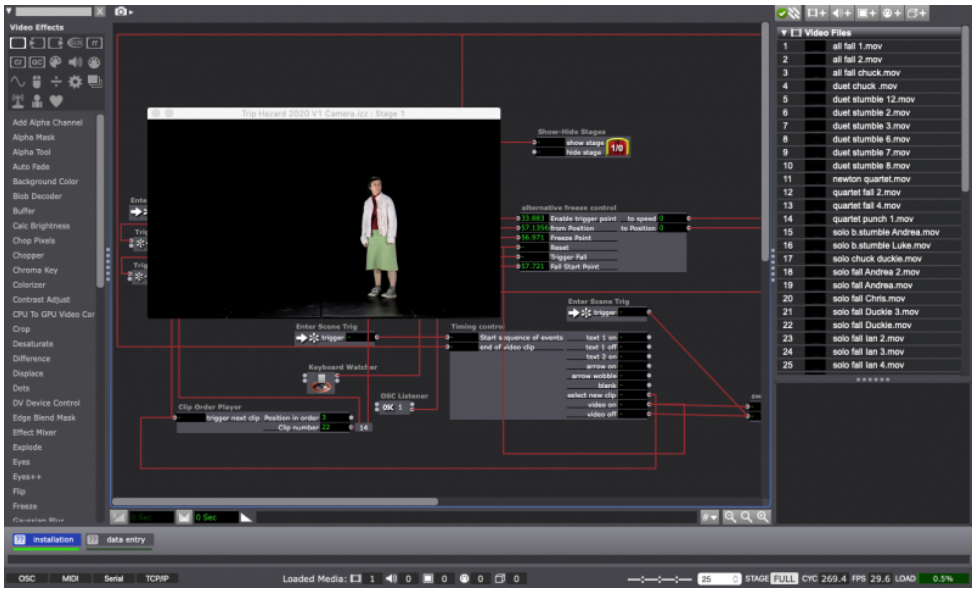


Figura 2.10: Ambiente de trabalho do *Isadora*

2.6.2 MaxMSP

MaxMSP é uma linguagem de programação visual desenvolvida e mantida pela *Cycling '74* para o desenvolvimento de um sistema de controlo e interação com diversos elementos multimédia. Pode ser dividida em duas componentes, *Max* é responsável pelas diversas comunicações no sistema, interface MIDI e com dispositivos de controlo e interação com o exterior. *MSP* é a componente responsável pelo processamento de sinal áudio e vídeo.

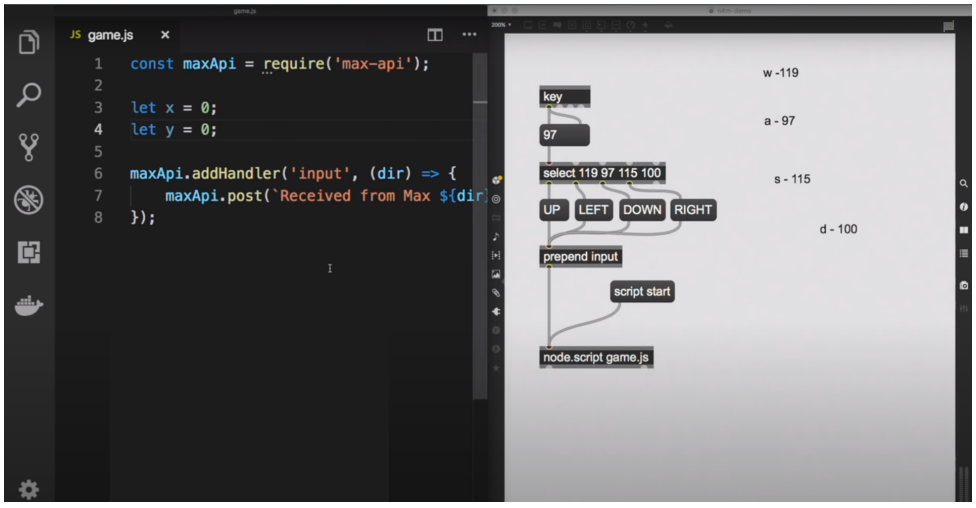


Figura 2.11: Ambiente de trabalho *MaxMSP*

Uma vez mais, baseia-se na interligação do objetos. Cada um destes objetos pode representar um componente de hardware, uma função para o processamento de sinal proveniente do objecto anterior ou uma interface de interação com o exterior.

Oferece uma utilização intuitiva para a conjugação dos vários objetos que compõem o sistema, mas por outro lado exige que o responsável pelo sistema tenha alguma capacidade e conhecimento de programação (dependendo sempre da função final do projeto) para o desenvolvimento das funcionalidades necessárias para cada objecto.

Relativamente ao custo, a subscrição deste software é cerca de 10€ por mês ou 100€ por ano[23].

3

Implementação

Tendo em conta os requisitos enumerados no capítulo 1 e as tecnologias selecionadas no capítulo 2 é, neste momento, importante descrever e especificar a totalidade do sistema desenvolvido.

Neste capítulo será descrita toda a metodologia e estratégia utilizada para o desenvolvimento e implementação de uma plataforma capaz de ultrapassar as dificuldades transmitidas pela Musa Paradisíaca.

O principal objectivo é expor temas relacionados com a arquitetura do sistema, linguagem de controlo desenvolvida, aplicação de controlo desenvolvida e configurações das diversas infraestruturas necessárias para a integração da solução implementada.

3.1 Arquitetura do sistema

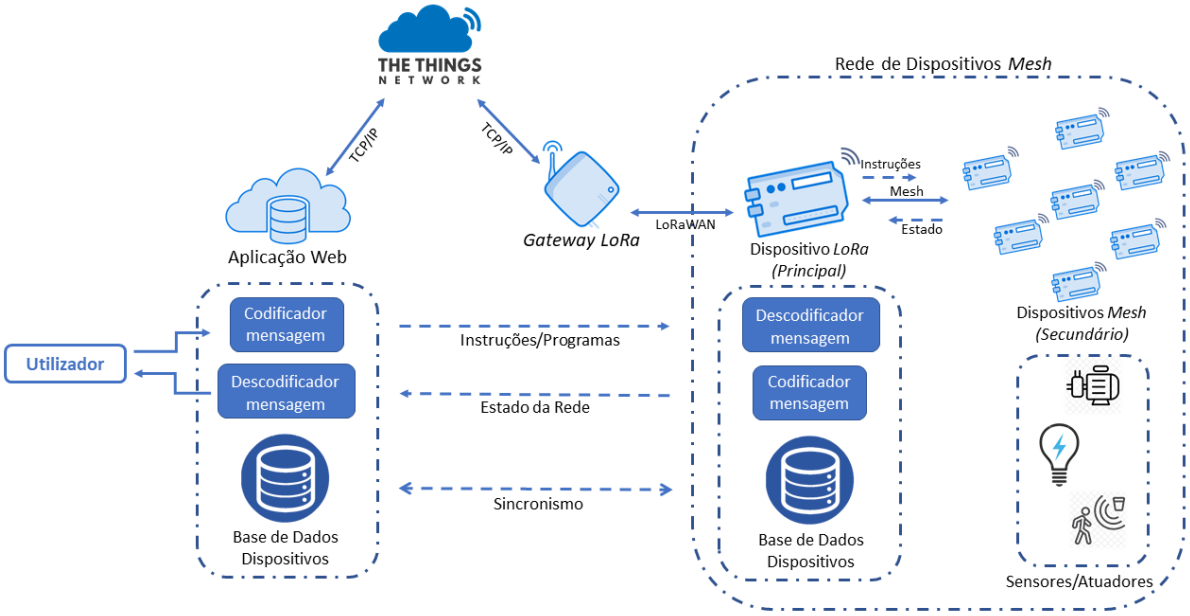


Figura 3.1: Arquitetura do Sistema

A figura 3.1 é uma representação da arquitetura do sistema implementado, onde estão presentes os diversos elementos que a compõem. Ao nível do *Hardware* resume-se, essencialmente, à utilização de dois tipos de dispositivos (Principal e Secundário) que, por sua vez, interagem com vários sensores/atuadores. A área do *software* engloba tudo aquilo que está relacionado com a codificação e descodificação das múltiplas mensagens transmitidas (tanto na aplicação *Web* como no dispositivo Principal), algoritmos de execução de comandos (dispositivo Secundário), *design* e configuração da aplicação *web*, configurações da infraestrutura de rede (TTN e *Gateway LoRa*).

Ao longo do presente capítulo cada um destes elementos será detalhado para melhorar a perceção sobre solução desenvolvida.

3.2 *Hardware*

A plataforma é composta por múltiplos dispositivos responsáveis pela interligação entre os diferentes elementos que compõem a instalação artística. Estes dispositivos têm a capacidade de interagir com os mais diversos elementos, desde a recolha e processamento de informação proveniente de sensores até à interação com relés/atuadores instalados nos diversos objetos artísticos.

Para esta função foram selecionados dois tipos de dispositivos:

- *TTGO T-Beam*
- *Esp32 Devkit v1*

Figura 3.2: *TTGO T-Beam*

Figura 3.3: *Esp32 Devkit v1*

Ambos partilham o mesmo micro controlador, mas têm funcionalidades diferentes fruto das capacidades presentes em cada um. O *TTGO T-Beam* funciona como dispositivo Principal, ao permitir a utilização do protocolo de comunicação LoRa é responsável pela interface entre a rede de dispositivos e a aplicação de gestão e controlo. Com o suporte da infraestrutura LoRaWAN o dispositivo Principal recebe as mensagens/-comandos provenientes da aplicação, descodifica as mesmas e retransmite-as para o(s) dispositivo(s) correspondente(s).

O *ESP32 Devkit* do ponto de vista hierárquico ocupa a posição de Secundário, recebe e executa as instruções recebidas do dispositivo Principal pode também comunicar com outros dispositivos Secundários se a instrução que receber assim o indicar.

O *TTGO T-Beam* utiliza a interface rádio de LoRa para comunicar com a aplicação e a interface de *Wi-Fi* para interagir, através do protocolo *ESP-Mesh*, com os vários *Esp32 Devkit* que compõem a rede de dispositivos.

3.2.1 Arquitetura Física

Na figura 3.4 está representado, de forma genérica e exemplificativa, o diagrama da arquitetura física implementada. Como descrito anteriormente o sistema é constituído por uma rede de dispositivos que comunicam entre si através do protocolo *ESP-Mesh*, e um dispositivo (Principal) que tem a capacidade de, em simultâneo, comunicar tanto com a rede *Mesh* como com a *gateway LoRa*. Neste dispositivo está implementado um algoritmo capaz de processar e endereçar as diferentes instruções/comandos para os vários elementos que compõem a rede *Mesh*. O facto do dispositivo Principal ser parte

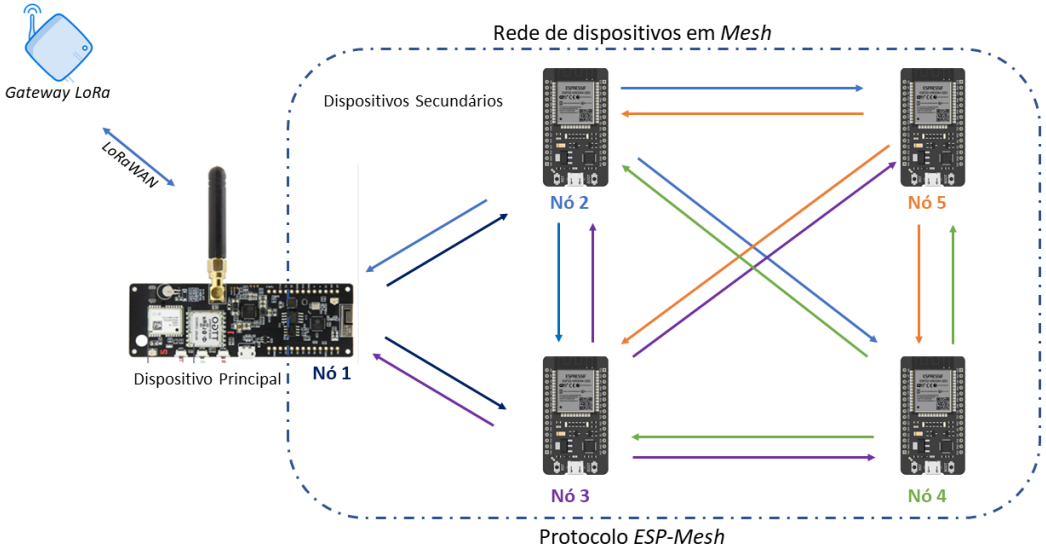


Figura 3.4: Arquitetura Física

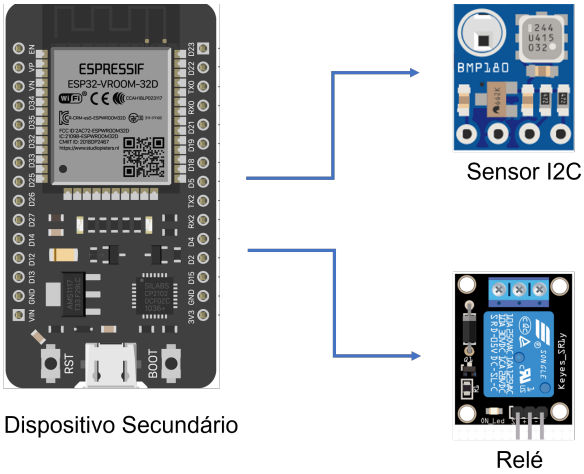


Figura 3.5: Dispositivo Secundário

integrante da rede *Mesh* oferece uma enorme vantagem para a implementação deste sistema ao possibilitar a comunicação com qualquer um dos elementos da rede (Secundários) de forma direta (se existir cobertura) ou de forma indireta (através de *multi-hop*). Este dispositivo pode, por isso, ser definido como o ponto de interface entre a instalação artística e o utilizador final da plataforma.

Importa salientar que cada dispositivo Secundário (3.5) pode interagir diretamente com os vários atuadores/relés e/ou sensores que compõem a instalação artística. Tendo em consideração este aspeto, foi desenvolvida uma interface de comunicação com sensores do tipo *I2C*, dada a elevada variedade deste tipo de sensores esta característica é bastante importante no contexto deste projeto, oferecendo ao sistema uma grande

versatilidade e capacidade de adaptação aos mais diversos cenários.

3.3 *Software*

Nesta secção será abordado todo o desenvolvimento de software necessário para a conjugação de todas as partes que compõem este sistema, quer seja através da parametrização de alguns elementos ou da elaboração e implementação de algoritmos nos mesmos. O sucesso e estabilidade na utilização da plataforma está diretamente relacionada com a eficiência dos diversos componentes de software presentes neste projeto.

3.3.1 Arquitetura Lógica

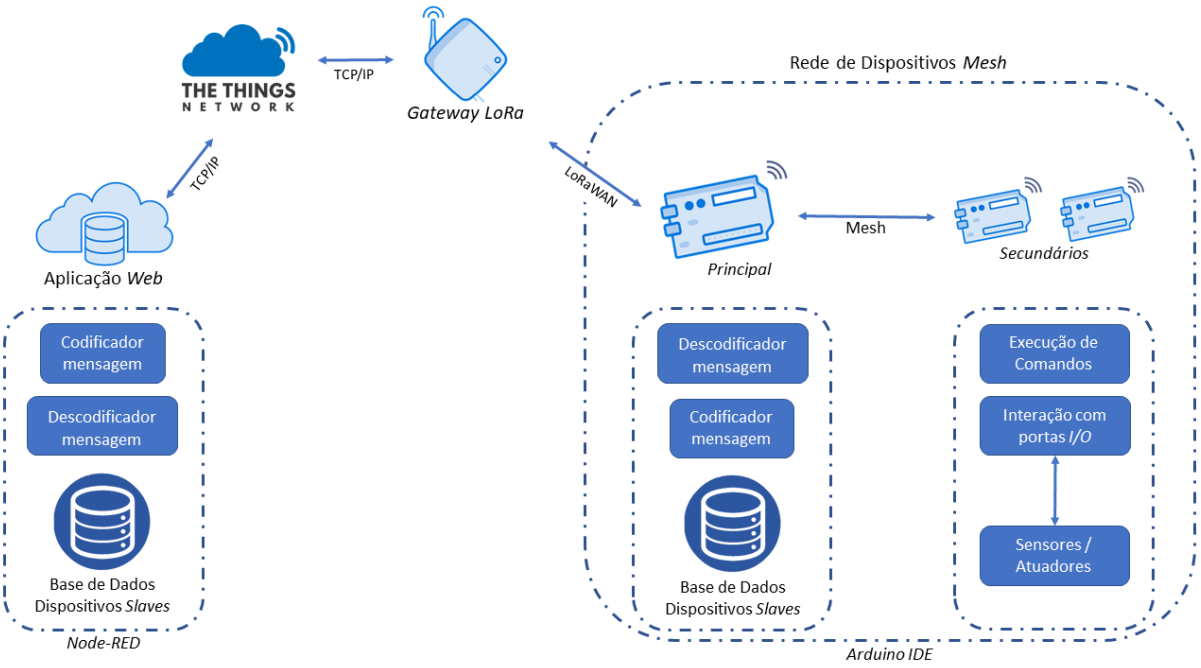


Figura 3.6: Arquitetura de *Software*

A figura 3.6 é uma representação de alto nível do software desenvolvido e implementado no contexto deste projeto. Seguindo a ordem presente na figura, na aplicação de interface com o utilizador o algoritmo desenvolvido em *Node RED* pode ser divido em três secções: codificação e transmissão dos comandos inseridos pelo utilizador, descodificação e representação da informação recebida da rede de dispositivos instalada e também implementação de uma base de dados (dinâmica) com todos os dispositivos da rede. Na infraestrutura de rede TTN e na *Gateway LoRa* não foi desenvolvido qualquer algoritmo, recorreu-se à parametrização destes dois elementos para possibilitar a

comunicação entre a aplicação e a rede de dispositivos.

Para a rede de dispositivos foram desenvolvidos dois softwares distintos. No dispositivo Principal o algoritmo implementado é dividido em 3 áreas: Descodificação e transmissão (para os dispositivos Secundário) dos comandos recebidos da aplicação, codificação e transmissão (para a aplicação) de diversos parâmetros de monitorização da rede e também a implementação (tal como na aplicação de interface com o utilizador) de uma base de dados com todos os dispositivos da rede. Uma vez que o dispositivo Principal possuí uma interface LoRa, o software implementado apresenta, também, uma componente de parametrização de forma a conseguir estabelecer comunicação com a infraestrutura de rede TTN.

No dispositivo Secundário o algoritmo pode ser dividido em 2 grandes áreas: Descodificação dos comandos recebidos (quer seja proveniente do Principal ou do Secundário) e a execução dos comandos recebidos. Este último pode implicar a interação com relés/atuadores ou a leitura e análise de dados provenientes de sensores conectados ao dispositivo.

Um aspeto comum nos dois tipos de dispositivo é a existência, no software, de uma área dedicada à configuração dos parâmetros associados ao protocolo *ESP-Mesh* para possibilitar a comunicação *Mesh* entre todos os elementos desta rede.

3.3.2 Armazenamento de Dados

Na implementação deste projeto o sincronismo entre a aplicação e o dispositivo Principal é fundamental para o correto funcionamento da plataforma, sem ele a interação entre o utilizador e a rede de dispositivos fica comprometida. O tópico "sincronismo"no contexto do projeto desenvolvido está, essencialmente, relacionado com uma condição: A aplicação e o dispositivo Principal têm de ter o mesmo conhecimento sobre que dispositivos Secundário compõem a rede, usando para isso um identificador único para cada dispositivo Secundário.

Para cumprir a condição descrita no parágrafo anterior foi implementada, na aplicação e no dispositivo Principal, uma base de dados para o registo de todos os dispositivos presentes na rede.

Na figura 3.7 estão representadas as duas bases de dados implementadas, "Dispositivos_app"na aplicação e a "Dispositivos_Mesh"no dispositivo Principal. Os campos "Id"e "Chip_id"(comuns em ambas as bases de dados) representam, respectivamente, o índice do dispositivo na base de dados (valor inteiro incrementado à medida que é adicionado um novo dispositivo) e o identificador único de cada dispositivo (valor

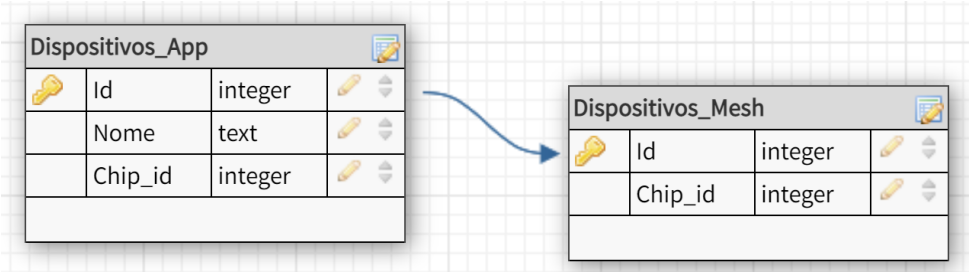


Figura 3.7: Base de Dados implementada

configurado de fábrica). Conforme representado (figura 3.7) existe uma correspondência entre os campos "id"de cada base de dados, ou seja, por exemplo: "id"3 do "Dispositivos_app"e o "id"3 do "Dispositivos_Mesh"têm o associado o mesmo o mesmo valor de "Chip_id".

O campo "Nome"surge apenas na base de dados implementada na aplicação e é preenchido com o nome atribuído pelo utilizador ao dispositivo adicionado, não sendo necessário para o funcionamento do dispositivo Principal.

Sempre que um dispositivo Secundário é adicionado à rede é necessário atualizar a base de dados existente na aplicação, de seguida é enviada (pela aplicação) uma mensagem com o "Id"e "Chip_id"para o dispositivo Principal atualizar a sua base de dados. Desta forma está garantido o sincronismo entre as duas bases de dados.

No próximo capítulo será abordado o impacto deste sincronismo na eficiência da comunicação entre a aplicação e a rede de dispositivos.

3.3.3 Linguagem Desenvolvida

Nesta secção será abordada a estratégia utilizada para o desenvolvimento da linguagem de comunicação entre a aplicação e o dispositivo Principal possibilitando, desta forma, a interação do utilizador com a instalação artística por ele implementada/idealizada.

Importa salientar que a estratégia aqui descrita é apenas referente ao bloco de informação útil presente em cada trama transmitida. O restante detalhe, relacionado com as configurações necessárias para estabelecer a comunicação entre os vários elementos, será abordado nos capítulos 3.3.5 e 3.3.6.

O grande desafio na idealização de uma linguagem de comunicação no contexto deste projeto, está diretamente relacionado com a tecnologia de longo alcance utilizada para a ligação entre a infraestrutura de rede TTN e a aplicação que, como descrito no capítulo 2.1.1, impõe algumas limitações tanto ao nível do tamanho de cada mensagem

enviada como quanto ao número de mensagens transmitidas. É, por isso, essencial implementar uma linguagem o mais eficiente possível de modo a obter o máximo rendimento de cada mensagem transmitida.

3.3.3.1 Formato da Mensagem

Com o intuito de melhorar a perceção sobre a estratégia utilizada, os diversos formatos de mensagem foram divididos em dois momentos/cenários:

- Mensagens trocadas entre a aplicação e o dispositivo Principal, ou seja, através do protocolo LoRaWAN.
- Mensagens trocadas na rede *Mesh* através do protocolo *ESP-Mesh*.

Mensagens enviadas da Aplicação para o dispositivo Principal - ***Downlink***:

1. Quando o utilizador necessita de adicionar um dispositivo na rede *Mesh* a mensagem transmitida tem o seguinte formato:

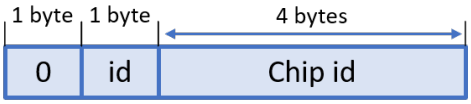


Figura 3.8: Adição de um novo dispositivo

   - 1º *byte* é preenchido com o valor "0";
   - 2º *byte* é preenchido com o id do dispositivo na base de dados da aplicação. Desta forma quando o dispositivo Principal atualizar a base de dados coloca o novo dispositivo Secundário na mesma posição que este se encontra na base de dados da aplicação.
   - Os quatro *byte*s seguintes (3º, 4º, 5º e 6º) são preenchidos com o *chip id* do novo dispositivo inserido. Este valor é colocado pelo fabricante e é único por dispositivo;

2. Quando o utilizador pretende enviar a lista comandos para os dispositivos da rede *Mesh* executarem a mensagem transmitida tem o seguinte formato:

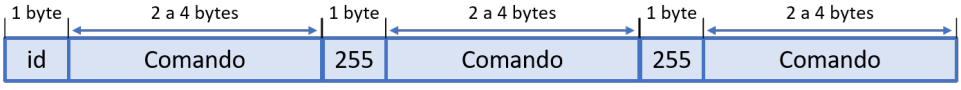


Figura 3.9: Envio da lista de comandos a executar

- 1º *byte* preenchido com o id do dispositivo (na base de dados da aplicação) a que se destina a lista de comandos. Como existe correspondência entre as duas base de dados (os dispositivos ocupam as mesmas posições nas respectivas bases de dados) não há necessidade de enviar o chip id (que ocupa 4 *byte*s) apenas o id do dispositivo na base de dados (que ocupa apenas 1 *byte*);
- Os *byte*s seguintes são ocupados com os vários comandos que o utilizador pretende enviar. Os comandos disponíveis apresentam diferentes codificações o que torna o seu tamanho variável. Entre comandos é colocado o valor "255"para servir de separador;

Mensagens enviadas do dispositivo Principal para a Aplicação - ***Uplink***:

1. Notificação que o dispositivo Principal recebeu corretamente a lista de comandos enviada pela aplicação:

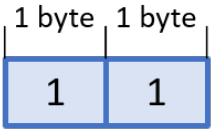


Figura 3.10: Receção da lista de comandos a executar

   - 1º *byte* é preenchido com o valor "1";
   - 2º *byte* é preenchido com o valor "1";

2. Notificação que o dispositivo Principal reencaminhou corretamente a lista de comandos para o seu destinatário na rede *Mesh*:

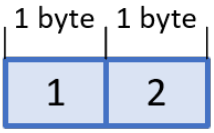


Figura 3.11: Reencaminhamento da lista de comandos

   - 1º *byte* é preenchido com o valor "1";
   - 2º *byte* é preenchido com o valor "2";

   Em condições de funcionamento normal estas duas mensagens são enviadas em conjunto, visto que o dispositivo Principal sempre que recebe uma lista de comandos reencaminha-a para o seu destinatário.

3. Notificação de quantos dispositivos estão ativos na rede *Mesh*:

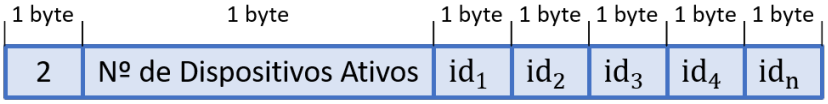


Figura 3.12: Indicação do número de dispositivos ativos

- 1º *byte* é preenchido com o valor "2";
- 2º *byte* é preenchido com o número de dispositivos ativos;
- Os *byte*s seguintes são preenchidos com os id's dos dispositivos (1 id/*byte*);

4. Notificação que o novo dispositivo adicionado pelo utilizador na aplicação, foi corretamente inserido na base de dados do dispositivo Principal:

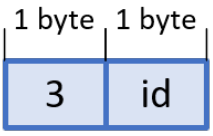


Figura 3.13: Novo dispositivo adicionado com sucesso

- 1º *byte* é preenchido com o valor "3";
- 2º *byte* é preenchido com o id do dispositivo inserido;

Mensagens trocadas entre os vários dispositivos da rede *Mesh* - **Intra Rede *Mesh***:

1. Quando o dispositivo Principal reencaminha a lista de comandos recebida para os destinatário correspondente a mensagem enviada tem o seguinte formato:

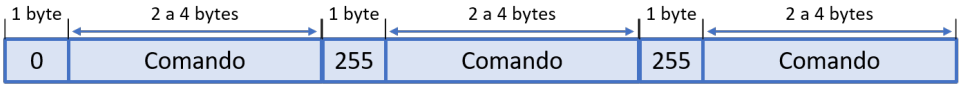


Figura 3.14: Envio da lista de comandos a executar

- 1º *byte* preenchido com o valor "0";
- Os *byte*s seguintes são ocupados com os vários comandos que o utilizador pretende enviar. Os comandos disponíveis apresentam diferentes codificações o que torna o seu tamanho variável. Entre comandos é colocado o valor "255"para servir de separador;

2. Quando, ao executar a lista de comandos recebida, o dispositivo Secundário tem a necessidade de interagir com outro dispositivo Secundário para que este atue sobre um determinado pino de *output* a mensagem enviada tem o seguinte formato:

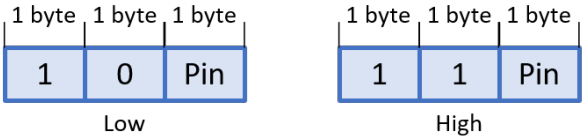


Figura 3.15: Interação com outro dispositivo Secundário

- 1º *byte* é preenchido com o valor "1";
- 2º *byte* é preenchido com o valor "0"ou "1"consoante a ação pretendida seja de "LOW"ou "HIGH"respetivamente;
- 3º *byte* é preenchido com o pino *output* sobre o qual se pretende atuar.

Resumo dos tipos de mensagens trocadas e os seus respetivos formatos:

Tabela 3.1: Mensagens trocadas em *Downlink*

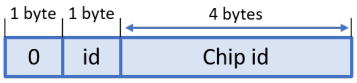
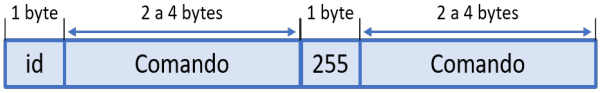
Tipo de Mensagem	Formato
Novo Sensor	
Lista de Comandos	

Tabela 3.2: Mensagens trocadas em *Uplink*

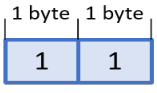
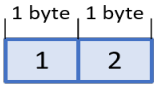
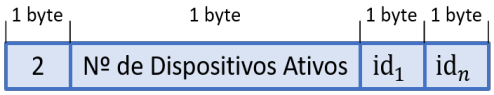
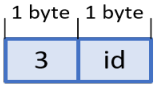
Tipo de Mensagem	Formato
Notificação de receção de comandos	
Reencaminhamento da lista de comandos	
Número de dispositivos ativos	
Dispositivos adicionado	

Tabela 3.3: Mensagens trocadas Intra Rede *Mesh* (Principal → Secundário)

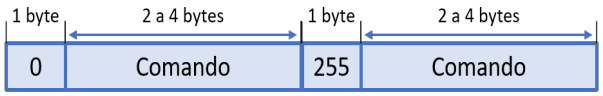
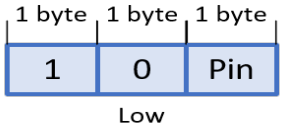
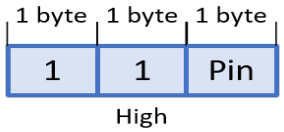
Tipo de Mensagem	Formato
Lista de Comandos	

Tabela 3.4: Mensagens trocadas Intra Rede *Mesh* (Secundário ↔ Secundário)

Tipo de Mensagem	Formato
Ação de "*Low*"	
Ação de "*High*"	

Na implementação deste projeto foram desenvolvidos diversos comandos com o intuito de oferecer ao utilizador a capacidade de programar a sua rede de dispositivos. Para cada comando foi definida uma sintaxe especifica, que tem de ser respeitada pelo utilizador para o correto funcionamento da aplicação e uma codificação também especifica para cada comando, utilizada sempre que existe a necessidade do envio de algum comando (figura 3.14 e 3.9). A forma como o utilizador interage com a aplicação para a escrita e envio dos comandos será abordada no capítulo 3.3.4.

Na tabela 3.5 apresentado um resumo de todos os comandos desenvolvidos e a respectiva funcionalidade.

Tabela 3.5: Lista de comandos implementadas

	Descrição
DstH (Pino, Destino)	Coloca a *"High"* o "Pino"do dispositivo "Destino"
DstL (Pino, Destino)	Coloca a *"Low"* o "Pino"do dispositivo "Destino"
ActionH (Pino)	Coloca a *"High"* o "Pino"do próprio dispositivo
ActionL (Pino)	Coloca a *"Low"* o "Pino"do próprio dispositivo
Sleep (x)	Coloca a execução em pausa "x"segundos
Var (Var, Valor)	Assigna a "Var"um "valor"
Add (Var, Valor)	Soma a "Var"um "valor"
Sub (Var, Valor)	Subtrai a "Var"um "valor"
Jmp (Linha)	Execução do programa salta para a "Linha"
JE (Var, Linha, Valor)	Se "Var"== "Valor", salta para a "Linha"
JNE (Var, Linha, Valor)	Se "Var"! = "Valor", salta para a "Linha"
JL (Var, Linha, Valor)	Se "Var"< "Valor", salta para a "Linha"
JLE (Var, Linha, Valor)	Se "Var"<= "Valor", salta para a "Linha"
JG (Var, Linha, Valor)	Se "Var"> "Valor", salta para a "Linha"
JGE (Var, Linha, Valor)	Se "Var">= "Valor", salta para a "Linha"
JZ (Var, Linha)	Se "Var"== 0, salta para a "Linha"
InputVAR (Var, Pino)	Assigna a "Var"a leitura no "Pino"
I2C_BMP280 (Var, Data)	Assigna a "Var"o tipo de leitura "Data"(temp, press e alt) do sensor BMP280

De seguida é exemplificado (com dois cenários distintos) de que forma é possível implementar os comandos enumerados na tabela 3.5.

Cenário 1:

- Arquitetura simples onde se pretende acionar o funcionamento de um LED ou atuador durante um período de 60 segundos.
- LED/Atuador ligado no pino 4 do Dispositivo1 Secundário.

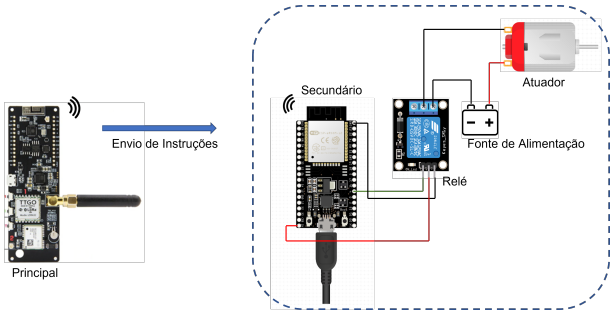


Figura 3.16: Arquitetura para o cenário 1

Uma das formas de realizar a tarefa descrita no cenário 1 está representada na figura 3.17, onde:

- **1ª linha** : *ActionH(4)* **-** coloca o pino 4 no estado "*High*";
- **2ª linha** : *Sleep(60)* **-** coloca a execução do programa em pausa durante 60 segundos;
- **3ª linha** : *ActionL(4)* **-** coloca o pino 4 no estado "*Low*".

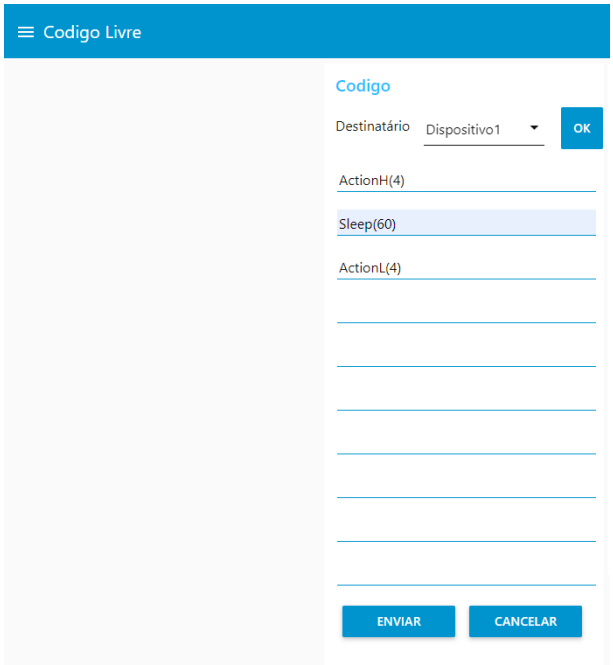


Figura 3.17: Código para o cenário 1

Cenário 2:

- Arquitetura mais complexa onde se pretende que o Dispositivo2 reaja (de forma cíclica) a uma ordem proveniente do Dispositivo1.
- Dispositivo1 recebe lista de comandos do Principal e está ligado a um sensor de temperatura.
- Se o valor de temperatura for superior a 22º o Dispositivo1 ordena o Dispositivo2 que accione um atuador.

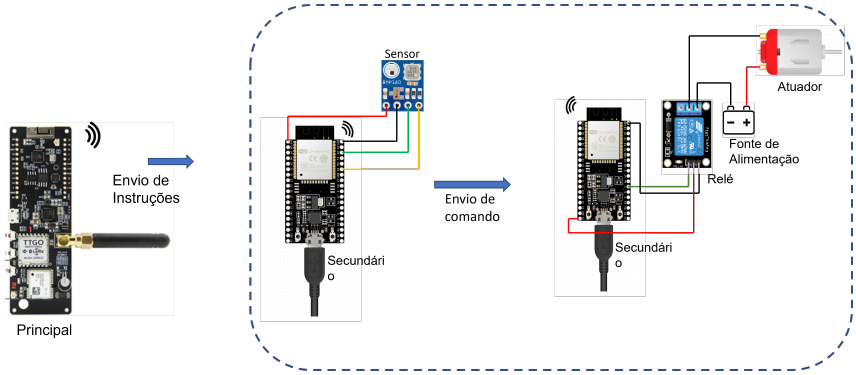


Figura 3.18: Arquitetura para o cenário 2

Uma das formas de realizar a tarefa descrita no cenário 2 está representada na figura 3.19, onde:

- **1ª linha** : *Var(1,0)* **-** inicia a variável1 com o valor "0";
- **2ª linha** : *DstL(4)* **-** coloca o pino 4 do Dispositivo2 no estado "*Low*";
- **3ª linha** : *I2C_BMP280(1,temp)* **-** assigna à variável1 o valor de temperatura, recolhido pelo sensor BMP280;
- **4ª linha** : *JLE(1,2,22)* **-** valida se o valor da variável1 é igual ou inferior a 22º, se esta condição se verificar a execução do programa passa para a 2ª linha, caso contrário continua a execução sequencial;
- **5ª linha** : *DstH(4)* **-** coloca o pino 4 do Dispositivo2 no estado "*High*";
- **6ª linha** : *Jmp(3)* **-** a execução do programa passa para a 3ª linha;

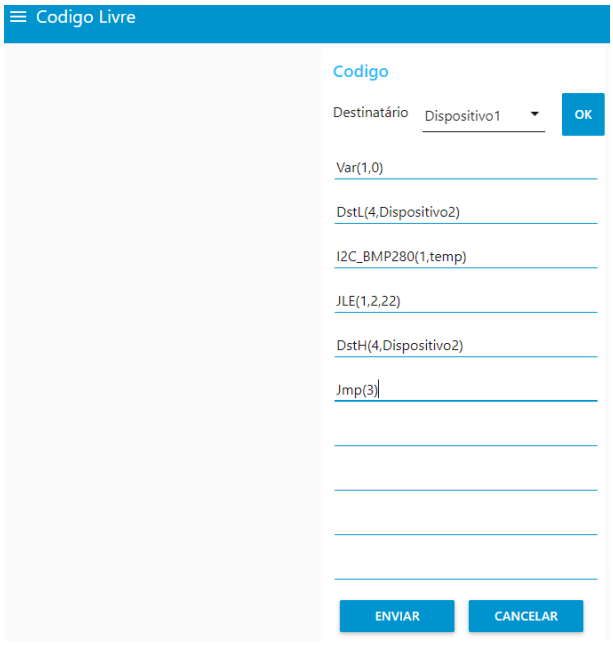


Figura 3.19: Código para o cenário 2

A codificação e sintaxe (de forma mais detalhada) implementada para cada um dos comandos foi a seguinte:

- **DstH (**Pino, Destino**)**:
  - **1º *byte*:** Preenchido com o valor "1";
  - **2º *byte*:** Preenchido com o pino *output* sobre o qual se pretende atuar;
  - **3º *byte*:** Preenchido com o "id"do dispositivo de destino;
  - **Descrição:** Comando utilizado para colocar o pino de um determinado dispositivo no estado "*High*". Utilizador coloca no primeiro parâmetro o pino que pretende afetar e no segundo o nome do dispositivo.
- **DstL (**Pino, Destino**)**:
  - **1º *byte*:** Preenchido com o valor "2";
  - **2º *byte*:** Preenchido com o pino *output* sobre o qual se pretende atuar;
  - **3º *byte*:** Preenchido com o "id"do dispositivo de destino;
  - **Descrição:** Comando utilizado para colocar o pino de um determinado dispositivo no estado "*Low*". Utilizador coloca no primeiro parâmetro o pino que pretende afetar e no segundo o nome do dispositivo.
- **Sleep (**Duração**)**:

  - **1º *byte*:** Preenchido com o valor "4";
  - **2º *byte*:** Preenchido com a duração em segundos.
  - **Descrição:** Comando utilizado para, durante a execução de um programa, temporizar uma determinada ação.

- **Var (**Var, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "5";
  - **2º *byte*:** Preenchido com a variável que se pretende afetar (de 1 a 5);
  - **3º *byte*:** Preenchido com o valor que se pretende colocar na variável.
  - **Descrição:** Comando utilizado para atribuir um valor a uma variável de memória. Para a implementação deste projeto foram criadas 5 variáveis de memória, ou seja, ao aplicar este comando o utilizador tem de selecionar uma das 5 variáveis criadas.

- **Add (**Var, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "6";
  - **2º *byte*:** Preenchido com a variável que se pretende afetar (de 1 a 5);
  - **3º *byte*:** Preenchido com o valor que se pretende somar à variável.
  - **Descrição:** Comando utilizado para somar à variável selecionada um determinado valor.

- **Sub (**Var, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "8";
  - **2º *byte*:** Preenchido com a variável que se pretende afetar (de 1 a 5);
  - **3º *byte*:** Preenchido com o valor que se pretende subtrair à variável.
  - **Descrição:** Comando utilizado para subtrair à variável selecionada um determinado valor.

- **Jmp (**Linha**)**:
  - **1º *byte*:** Preenchido com o valor "7";
  - **2º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **Descrição:** Comando utilizado para alterar a sequência normal de execução de um programa.

- **JE (**Var, Linha, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "11";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se os valores forem iguais a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.
- **JLE (**Var, Linha, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "12";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se os valores forem iguais ou o valor for inferior à variável a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.
- **JNE (**Var, Linha, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "13";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se os valores

forem diferentes a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.

- **JG (**Var, Linha, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "14";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se o valor for superior à variável a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.
- **JGE (**Var, Linha, Valor**)**:
  - **1º *byte*:** Preenchido com o valor "15";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se os valores forem iguais ou o valor for superior à variável a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.
- **JZ (**Var, Linha**)**:
  - **1º *byte*:** Preenchido com o valor "16";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;

  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor "0". Se o valor da variável for igual a "0"a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.

- **JL (**Var, Linha, Valor**)**:

  - **1º *byte*:** Preenchido com o valor "17";
  - **2º *byte*:** Preenchido com a variável que se pretende consultar;
  - **3º *byte*:** Preenchido com a linha para a qual se pretende que a execução do programa continue;
  - **4º *byte*:** Preenchido com o valor utilizado na comparação;
  - **Descrição:** Comando utilizado para realizar uma decisão com base numa operação aritmética. Comparação entre o valor guardado na variável passada no primeiro parâmetro e o valor do terceiro parâmetro. Se o valor for inferior à variável a sequência do programa continua na linha passada no segundo parâmetro. Caso a condição não se verifique o programa continua a sua sequência normal.

- **ActionH (**Pino**)**:

  - **1º *byte*:** Preenchido com o valor "10";
  - **2º *byte*:** Preenchido com o pino *output* sobre o qual se pretende atuar;
  - **Descrição:** Comando utilizado para colocar o pino do dispositivo (a que se destina o programa) no estado *"High"*.

- **ActionL (**Pino**)**:

  - **1º *byte*:** Preenchido com o valor "9";
  - **2º *byte*:** Preenchido com o pino *output* sobre o qual se pretende atuar;
  - **Descrição:** Comando utilizado para colocar o pino do dispositivo (a que se destina o programa) no estado *"Low"*.

- **InputVar (**Var, Pino**)**:

  - **1º *byte*:** Preenchido com o valor "18";
  - **2º *byte*:** Preenchido com a variável que se pretende afetar (de 1 a 5);

  - **3º *byte*:** Preenchido com o pino de *input* sobre o qual se pretende fazer a leitura;
  - **Descrição:** Comando utilizado para afetar a variável passada no primeiro parâmetro com a leitura (*digitalRead*) efetuada no pino passado no segundo parâmetro .

- **I2C_BMP280 (**Var, Info**)**:
  - **1º *byte*:** Preenchido com o valor "19";
  - **2º *byte*:** Preenchido com a variável que se pretende afetar (de 1 a 5);
  - **3º *byte*:** Preenchido com o tipo de informação que se pretende consultar (Temperatura, Pressão, Altitude);
  - **Descrição:** Comando utilizado para interagir com um sensor do tipo I2C (neste caso o BMP 280). Assigna à variável passada no primeiro parâmetro a leitura efetuada pelo sensor, de acordo com a informação passada no segundo parâmetro.

Estas codificações são utilizadas no campo "comando"representado nas mensagens descritas nas tabelas 3.1 e 3.3.

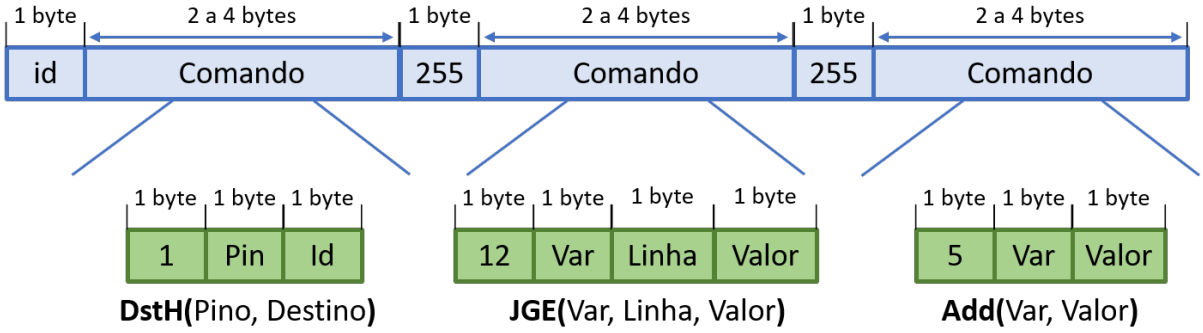


Figura 3.20: Exemplo para envio de uma mensagem

Na figura 3.20 está exemplificado a forma como o conteúdo do campo "comando"é preenchido com a codificação de cada comando enviado numa mensagem. O exemplo em causa usa o formato da mensagem trocada em *downlink* mas aplica-se de igual forma para as mensagens trocadas intra rede (à exceção do primeiro *byte*).

Nos próximos capítulos será abordada os diversos métodos de codificação e descodificação das várias mensagens trocadas no sistema.

3.3.3.2 Algoritmo de codificação

Uma vez expostos os vários formatos de mensagens transmitidas é agora importante descrever o algoritmo de codificação desenvolvido.

Sempre que é preciso transmitir uma mensagem é necessário recorrer a um processo de codificação, como explicado na secção anterior, a transmissão de mensagens ocorre da aplicação para a rede de dispositivos (e vice-versa) e de dispositivo para dispositivo.

Foi, por isso, implementado um algoritmo de codificação tanto ao nível da aplicação de interface com o utilizador como dos dispositivos que compõem a rede.

Ao nível da aplicação existem dois momentos em que o algoritmo de codificação é utilizado:

- Quando o utilizador pretende adicionar um novo dispositivo à rede implementada.
- Quando o utilizador pretende enviar um programa para a rede de dispositivos executar.

Para o primeiro caso, o utilizador no campo "Nome do Dispositivo"o nome pelo qual pretende identificar o dispositivo e no campo imediatamente abaixo coloca o *chip ID* (número identificativo proveniente de fábrica).

Figura 3.21: Aplicação - Nome e *Chip ID*

Ao colocar estas duas informações a base de dados da aplicação é atualizada com a informação inserida e é associado um id (na base de dados) ao novo dispositivo. Ao clicar no botão "Adicionar Dispositivo"é iniciado o processo de codificação da mensagem a enviar. Conforme apresentado na figura 3.8, o algoritmo faz a concatenação entre o id atribuído ao novo dispositivo com o *chip ID* adicionando, também, um *byte* com um valor identificativo do tipo de mensagem. Através do protocolo de interface

entre o *Node-RED* e o TTN a mensagem é transmitida para a rede de dispositivos, mais concretamente para o dispositivo Principal.

Na situação em que o pretendido é enviar um programa para a rede de dispositivos executar, o utilizador seleciona qual o destinatário do programa desenvolvido e de seguida preenche cada uma das linhas com o comando pretendido (de acordo com a sintaxe apresentada na tabela 3.5)

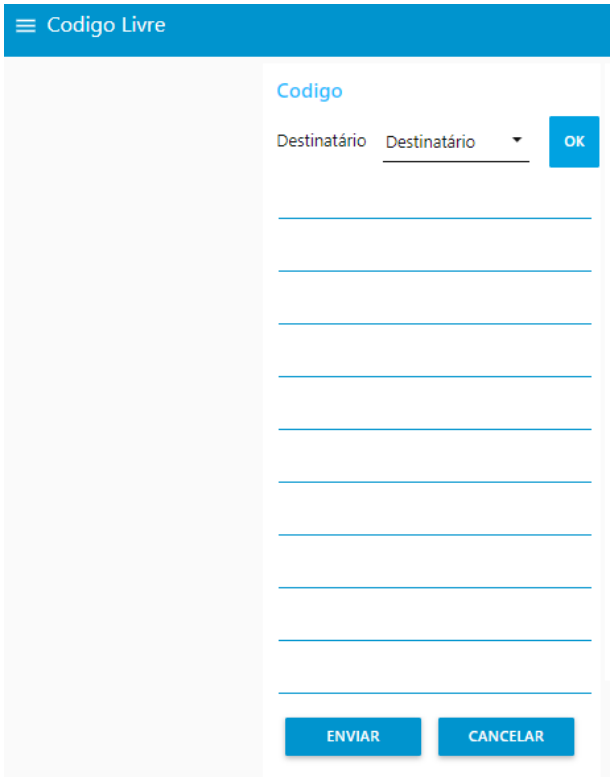


Figura 3.22: Aplicação - Destinatário e Comandos

Ao selecionar o nome do destinatário, é feito um *query* à base de dados de forma a obter o id correspondente. De seguida, as linhas preenchidas são codificadas dependendo do comando escrito pelo utilizador. É então feita a concatenação do id do dispositivo com os vários comandos codificados e entre cada comando é colocado um separador como demonstrado na figura 3.20. Com o processo de codificação concluído (e depois de clicar no botão "Enviar") a mensagem é enviada para o dispositivo Principal, uma vez mais através da interface com o TTN.

Ao nível dos dispositivos existem três cenários em que o processo de codificação é utilizado:

- Quando o dispositivo Principal necessita de enviar alguma informação para a aplicação.

- Quando o dispositivo Principal necessita de reencaminhar um programa (recebido da aplicação) para um determinado Secundário.

- Quando um dispositivo Secundário tem de comunicar com outro dispositivo.

Para o primeiro cenário o processo de codificação é relativamente simples, conforme demonstrado nas figuras 3.10, 3.11 e 3.13, é a concatenação de dois *byte*s em que o 1º é um valor indicativo do tipo de mensagem e o 2º varia dependendo do tipo de mensagem. Na situação representada na figura 3.12 o processo é ligeiramente mais complexo, sempre que existe uma alteração na rede de dispositivos é feito um ciclo de *queries* à base de dados no dispositivo Principal para retornar o id dos dispositivos que estão ativos na rede *mesh*, sendo também possível obter a informação de quantos dispositivos estão, naquele momento, ativos na rede. Desta forma está recolhida toda a informação para a composição da mensagem representada na figura 3.12.

Para o segundo cenário, o processo de codificação inicia-se assim que o dispositivo Principal recebe (da aplicação) uma lista de comandos para a rede executar. A primeira fase é a identificação do id do dispositivo destinatário da lista de comandos, com essa identificação concluída é feita uma *query* à base de dados para retornar o *chip ID* correspondente. De seguida é feita uma análise à lista de comandos recebida e se existir algum comando que tenha como destinatário outro dispositivo, como por exemplo: DstH ou DstL (tabela 3.5), o id passado no argumento é convertido para o *chip ID* correspondente (figura 3.23).

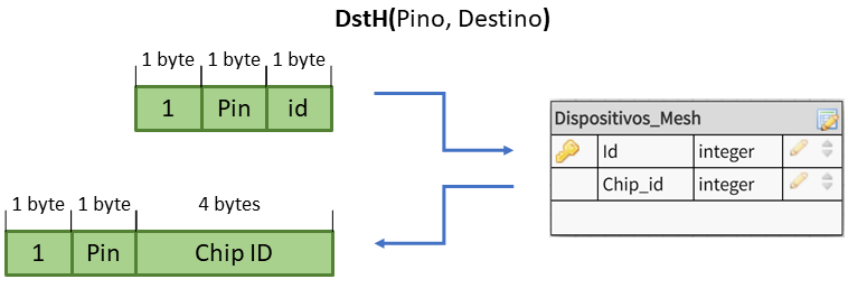


Figura 3.23: Conversão de id para *chip ID*

Uma vez concluído esta análise o dispositivo Principal envia a mensagem ao destinatário constituída pela concatenação dos comandos recebidos (com um separador entre cada um) com, uma vez mais, um *byte* inicial com o valor indicativo do tipo de mensagem a transmitir (conforme representado na figura 3.24)

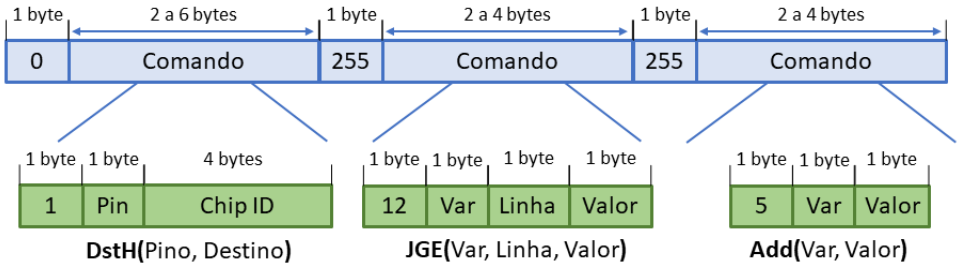


Figura 3.24: Exemplo para envio de uma mensagem intra rede

O 3º cenário, surge quando a meio da execução de um programa o dispositivo Secundário em causa tem a necessidade de interagir com outro dispositivo Secundário solicitando a mudança de estado de um determinado pino. O processo de codificação neste caso também é relativamente simples, é a concatenação de 3 *byte*s (de acordo com a representação na figura 3.15) , em que o 1º é indicativo do tipo de mensagem transmitida, o 2º varia conforme o pedido seja de "*High*"ou "*Low*"e o 3º é a informação de qual pino afetar.

3.3.3.3 Algoritmo de descodificação

Sempre que algum dispositivo ou a aplicação recebem uma mensagem é necessário recorrer a um processo de descodificação de modo a atuar em função da mensagem recebida. Foi, por isso, desenvolvido um algoritmo de descodificação tanto ao nível da aplicação como ao nível dos diversos dispositivos.

Ao nível dos dispositivos existem 4 tipos de mensagens recebidas (2 pelo dispositivo Principal e 2 pelos dispositivos Secundários):

- Principal:
  - Mensagem com o novo dispositivo adicionado à rede *mesh*.
  - Mensagem com a lista de comandos a enviar para a rede *mesh*.
- Secundário
  - Mensagem com a lista de comandos a executar.
  - Mensagem recebida de outro Secundário com ação a executar.

A primeira ação realizada em todos os processos de descodificação (tanto na aplicação como nos dispositivos) é a leitura do 1º *byte* da mensagem recebida, o valor do primeiro *byte* define o tipo de mensagem recebido (como se constata no capítulo 3.3.3.1) o que desencadeia processos de descodificação distintos.

No primeiro cenário exposto, o algoritmo de descodificação tem como função atualizar a base de dados do dispositivo Principal com o novo dispositivo adicionado. No 2º *byte* está a posição da base de dados que deve ser atualizada com o *chip ID* do novo dispositivo. O *chip ID* está dividido entre os 3º, 4º, 5º e 6º *byte*s recorre-se, por isso, a uma operação aritmética para converter 4 valores de 1 *byte* para 1 valor de 4 *bytes*.

De seguida o id da base de dados do dispositivo Principal é atualizado cm o *chip ID* recebido, desta forma, existe sempre uma correspondência entre a base de dados da aplicação com a do dispositivo Principal.

Para o segundo cenário sempre que o dispositivo Principal recebe uma lista de comandos o processo de descodificação inicia-se com a identificação do destinatário da lista de comandos, de seguida os comandos codificados na mensagem são separados por linhas em que cada linha corresponde a um comando (isto só é possível porque, na mensagem recebida, entre cada comando existe um separador). Com este processo concluído desencadeia-se o processo de codificação explicado na secção anterior que torna possível o envio da lista de comandos para o respectivo destinatário.

Ao contrário dos casos anteriores, em que as mensagens são recebidas no dispositivo Principal no 3º cenário a mensagem é recebida pelo dispositivo Secundário. Ao receber uma lista de comandos o objectivo do dispositivo Secundário é executar o programa recebido. A primeira tarefa realizada pelo algoritmo de descodificação é organizar e estruturar a mensagem recebida em forma de lista numerada em que cada linha é preenchida com um comando. Com esta tarefa concluída o passo seguinte é a análise de cada comando linha a linha e de forma sequencial. Conforme explicado no capítulo 3.3.3.1 o 1º *byte* de cada comando é o fator de diferenciação entre cada comando (tabela 3.5), desta forma o algoritmo executa diferentes funções dependendo do comando em causa. Esta análise é feita linha a linha até ao fim do programa, mas o facto de estarem implementados diversos comandos de *"jumps"* para outras linhas abre, também, a possibilidade do programa recebido não ter fim, ou seja o algoritmo pode ficar num ciclo de execução de forma indeterminada no tempo, tudo depende do programa implementado pelo utilizador.

Para o último cenário o processo de descodificação é mais simples que os expostos anteriormente, conforme ilustrado na figura 3.15 do 2º *byte* retira-se a informação sobre o estado que se pretende colocar um determinado pino (*High* ou *Low*) e do 3º *byte* retira-se qual o pino a afetar. Com esta informação recolhida o processo de descodificação é concluído com a execução da função *digitalWrite()* que permite atuar sobre o pino pretendido.

Ao nível da aplicação existem 4 momentos em que o processo de descodificação é utilizado:

- Quando o dispositivo Principal notifica a aplicação que recebeu com sucesso a lista de comandos.
- Quando o dispositivo Principal notifica a aplicação que reencaminhou a lista de comandos para o dispositivo Secundário destinatário.
- Quando o dispositivo Principal notifica a aplicação que o novo dispositivo foi inserido corretamente na base de dados.
- Quando o dispositivo Principal envia a lista de dispositivos ativos na rede.

O processo de descodificação, à semelhança do que nos dispositivos, começa sempre pela análise do 1º *byte*. Para as primeiras duas situações a aplicação recebe uma mensagem composta por 2 *bytes* (figura 3.10 e 3.11), o processo de descodificação é concluído com a execução de um *pop-up* com a mensagem "Lista de comandos recebida com sucesso"(no caso do 1º *byte* ser o valor "1") ou com a mensagem "Lista de comandos enviada para a rede de dispositivos"(no caso do 2º *byte* ser o valor "2").

Na terceira situação o processo de descodificação é um pouco mais complexo, utilizando o id extraído no 2º *byte* da mensagem recebida (3.13), é feito um *query* à base de dados para perceber qual o nome associado ao id enviado pelo dispositivo Principal. O processo termina com a execução, uma vez mais, de um *pop-up* com a mensagem "(nome do dispositivo) inserido na base de dados do dispositivo Principal"desta forma o utilizador tem a confirmação que o dispositivo por ele adicionado foi o mesmo que o dispositivo Principal enviou.

Para a quarta situação o tamanho da mensagem recebida varia consoante o número de dispositivos ativos (3.12) o processo de descodificação inicia-se com a identificação do número de dispositivos ativos na rede (2º *byte*), com este conhecimento já adquirido o algoritmo de descodificação já sabe quantos id's espera receber e quantos *bytes* tem de descodificar. Por cada id recebido é feito um *query* à base de dados para retornar o nome do dispositivo. O processo concluí-se com a atualização da tabela dos dispositivos ativos na rede *mesh*.

3.3.4 Aplicação de Gestão e Controlo da rede *Mesh*

Esta secção incidirá sobre a estratégia implementada para o desenvolvimento e *design* da aplicação de interface com o utilizador utilizando o *Node-RED*, assim como a explicação detalhada dos vários menus e sub-menus desenvolvidos na aplicação. Será também abordada todas as funcionalidades e instruções de operação da aplicação desenvolvida.

3.3.4.1 Desenvolvimento e *Design*

O objetivo principal desta aplicação é servir de interface entre o utilizador e a rede *mesh* de dispositivos e divide-se em duas áreas principais: a área de gestão e controlo de dispositivos e a área destinada ao envio de comandos para a rede de dispositivos.

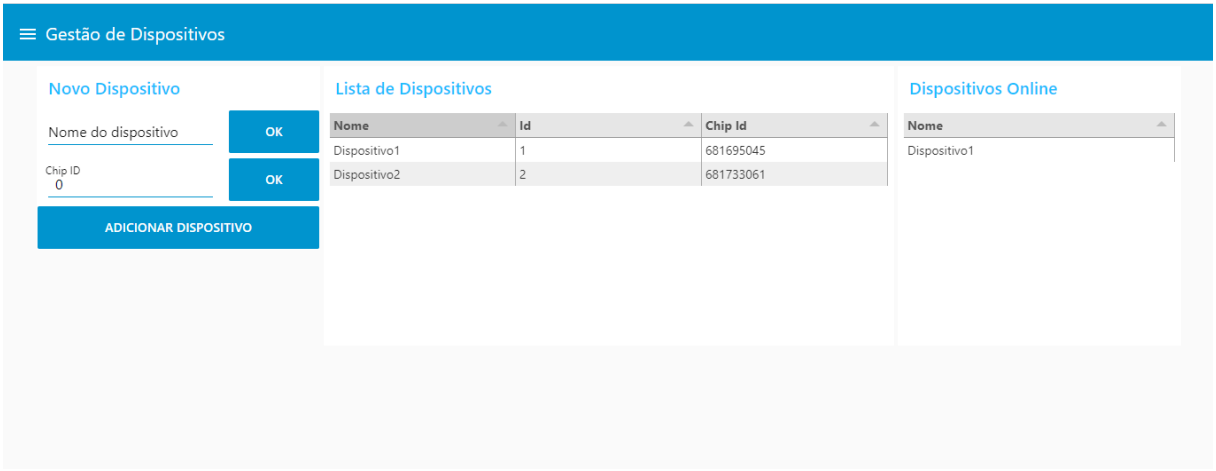


Figura 3.25: Pagina inicial da aplicação

Assim que a aplicação é inicializada, por definição, a primeira área apresentada (figura 3.25) diz respeito ao controlo e gestão de dispositivos. Neste separador da aplicação o utilizador tem a possibilidade de:

- Registar um novo dispositivo na rede *mesh*;
- Consultar a lista de dispositivos registados na rede *mesh*;
- Consultar quais os dispositivos *online* na rede *mesh*

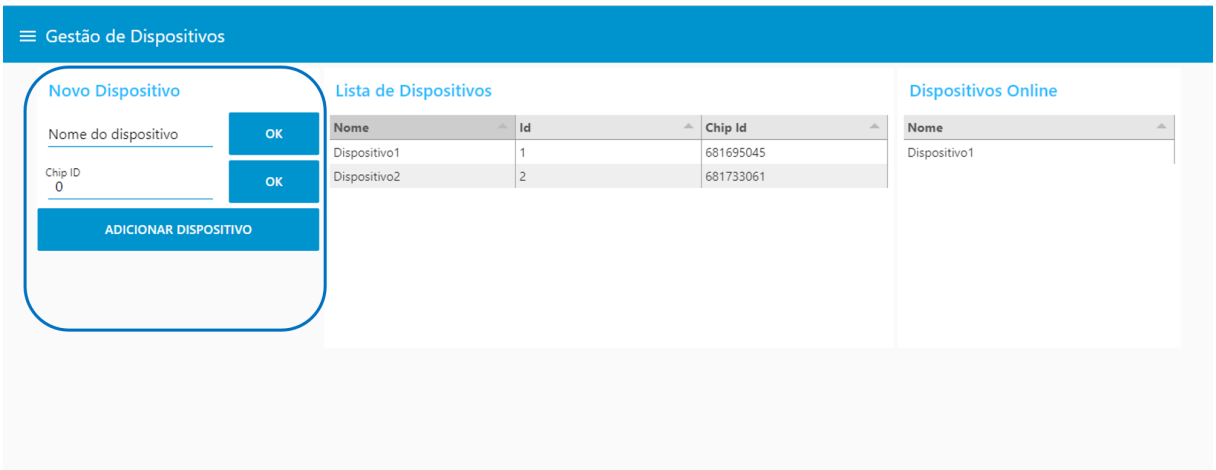


Figura 3.26: Registo de um novo dispositivo

A zona identificada na figura 3.26, destina-se ao registo de um novo dispositivo. É colocado o nome do dispositivo (que será usado como identificação do dispositivo) e no campo abaixo o *chip ID* fornecido pelo fabricante. Para concluir o registo do novo dispositivo o utilizador clica no botão "Adicionar Dispositivo". Internamente aquilo que acontece é a atualização da base de dados com os dados inseridos pelo utilizador. Assim que o utilizador confirma os dados inseridos (ao clicar no botão) é enviado uma mensagem para a rede *mesh* com a informação do novo dispositivo (como explicado no capítulo 3.3.3.1).

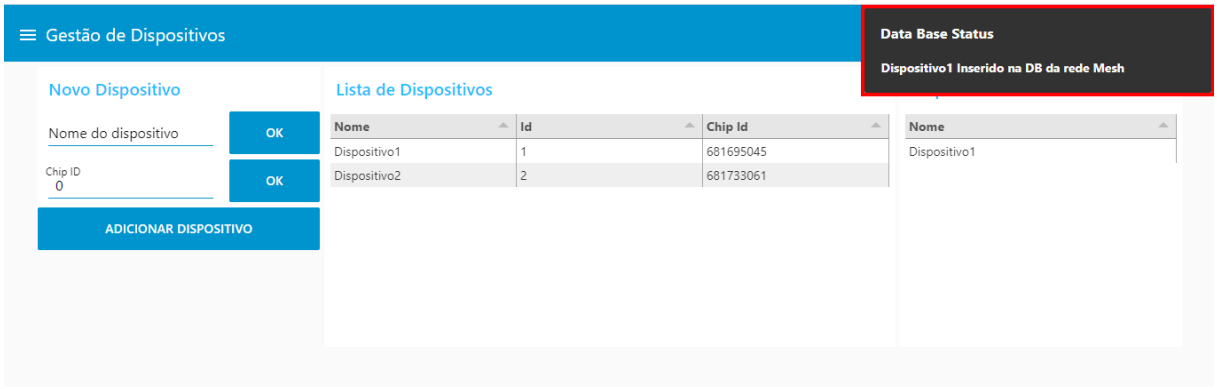


Figura 3.27: *Pop-up* de notificação do registo na base de dados

Com o intuito de informar o utilizador sobre o sucesso da adição do dispositivo na rede foi implementado um *pop-up* com essa informação (figura 3.27).

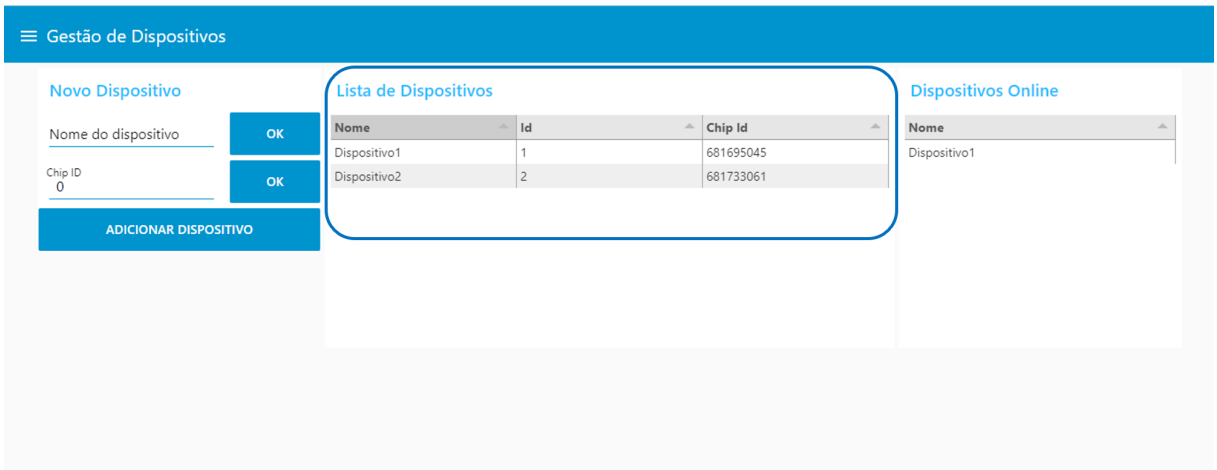


Figura 3.28: Lista de dispositivos

A zona evidenciada na figura 3.28 é a lista de todos os dispositivos inseridos pelo utilizador sendo, por isso, um espelho da base de dados implementada na aplicação. Com este quadro o utilizador tem informação do nome, *chip ID* e id na base de dados

dos varios dispositivos. Desta forma o utilizador tem uma visão sobre que dispositivos são do conhecimento da rede *mesh*.

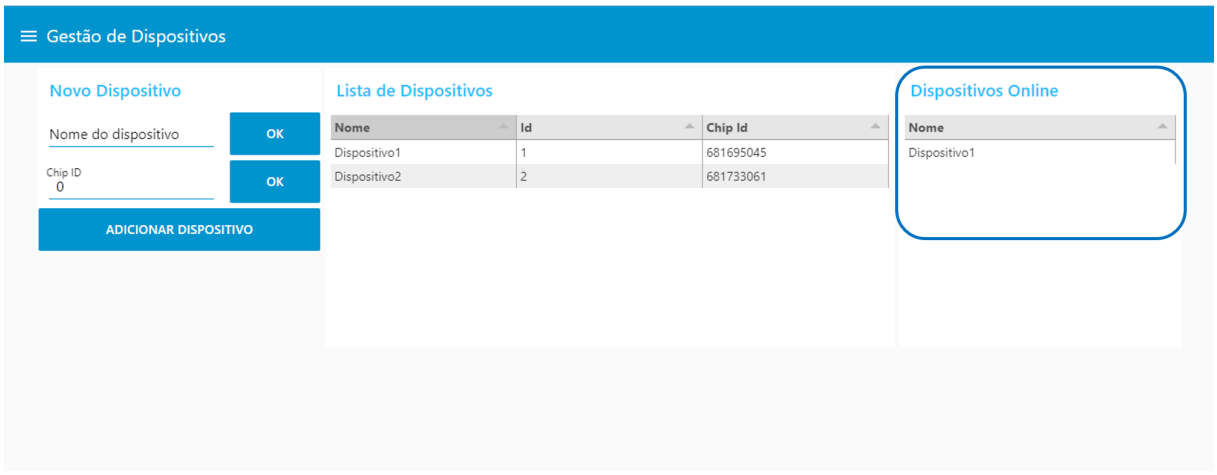


Figura 3.29: Lista de dispositivos *online*

A ultima zona desenvolvida neste separador (figura 3.29) é a lista de dispositivos *online* na rede *mesh*. Esta informação torna-se particularmente útil para o utilizador perceber quais os dispositivos operacionais na rede *mesh*. Esta lista é atualizada de forma dinâmica, sempre que existe uma alteração na rede (mudança de estado de um dispositivo) a aplicação recebe uma mensagem com os dispositivos *online* (capítulo 3.3.3.1) e a lista é atualizada.

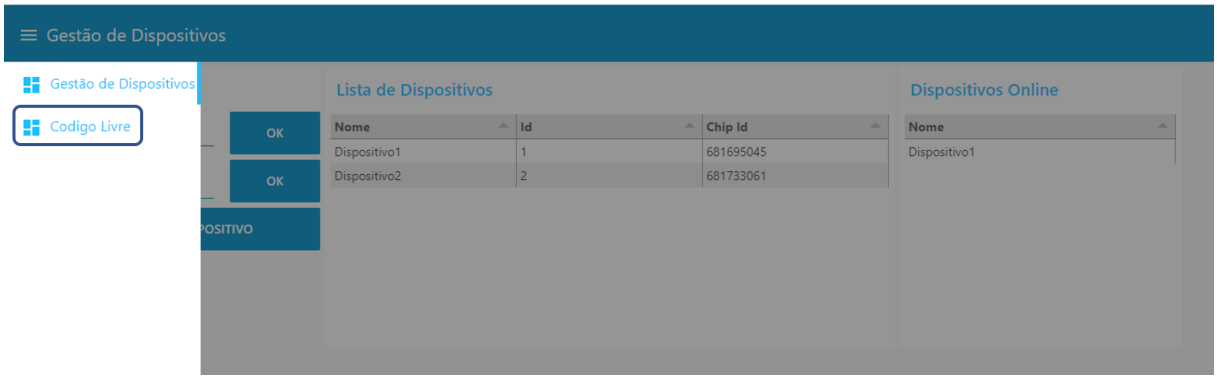


Figura 3.30: Troca de separador

Para aceder à área destinada ao envio de uma lista de comandos para a rede *mesh*, conforme explicado na figura 3.30 seleciona-se a secção "Código Livre". Neste separador da aplicação o utilizador tem a possibilidade de:

- Instruir a rede *mesh* a executar uma determinada ação através do envio de comandos;

- Consultar a lista de comandos disponível assim como a respectiva descrição e funcionalidade;
- Consultar as conexões implementadas entre os dispositivos Secundário.

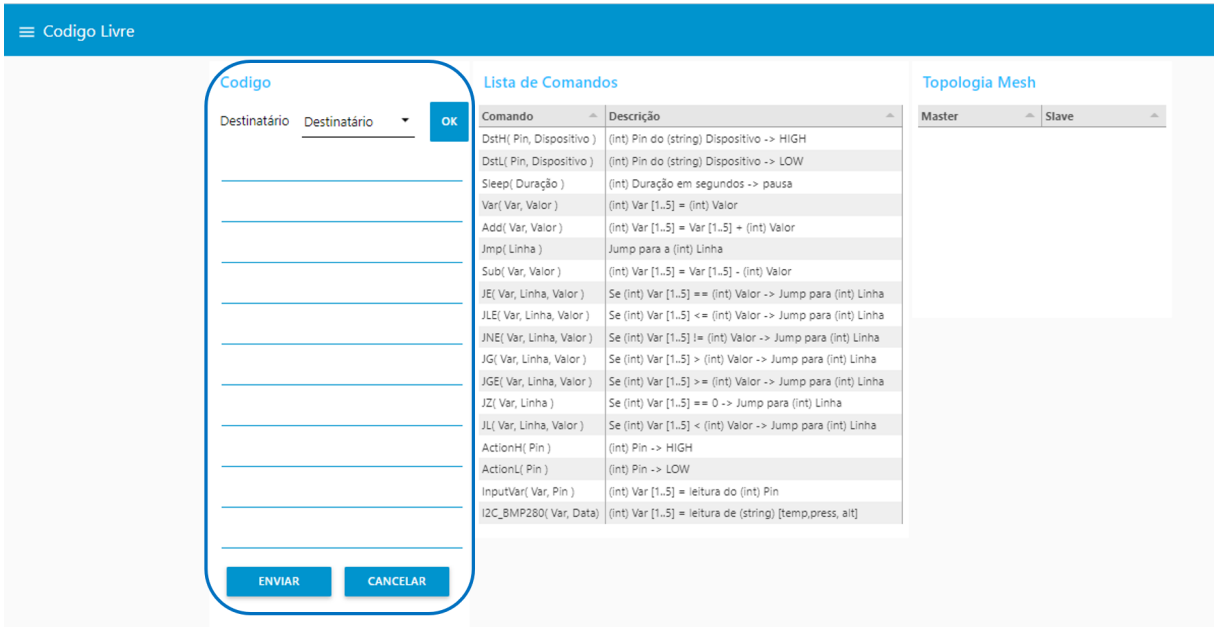


Figura 3.31: Implementação do código

A primeira zona (identificada na figura 3.31) é o local onde o utilizador escreve os diversos comandos de forma a instruir a rede *mesh*. O utilizador começa por selecionar, de entre as varias opções, o destinatário da lista de comandos (figura 3.32).

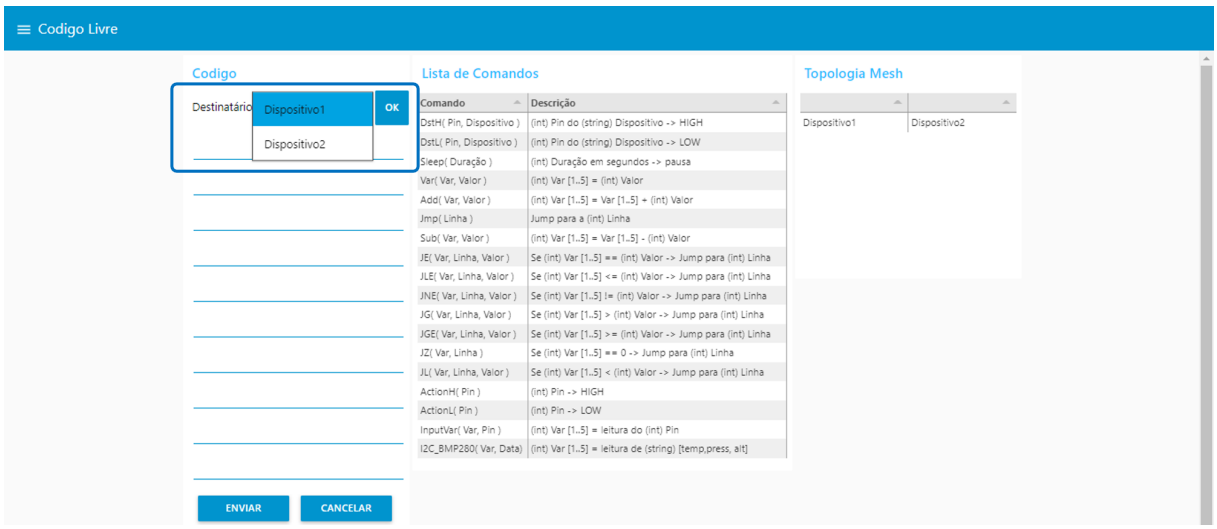


Figura 3.32: Seleção do destinatário

De seguida coloca um comando por cada linha disponibilizada (sempre de acordo com a sintaxe apresentada no quadro 3.5). Com o programa concluído é necessário clicar no

botão "Enviar", para que a lista de comandos seja enviada para a rede *mesh* (conforme explicado no capítulo 3.3.3.1).

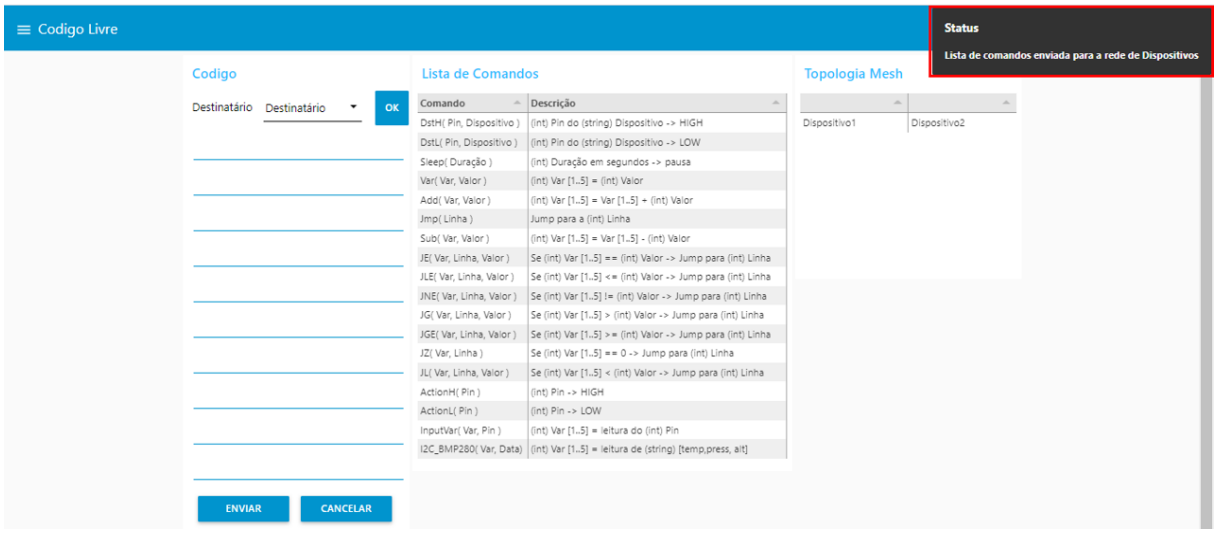


Figura 3.33: *Pop-up* de notificação do envio da lista de de comandos

Com o intuito de informar o utilizador sobre o sucesso da entrega do programa desenvolvido, surge um *pop-up* com essa informação (figura 3.33).

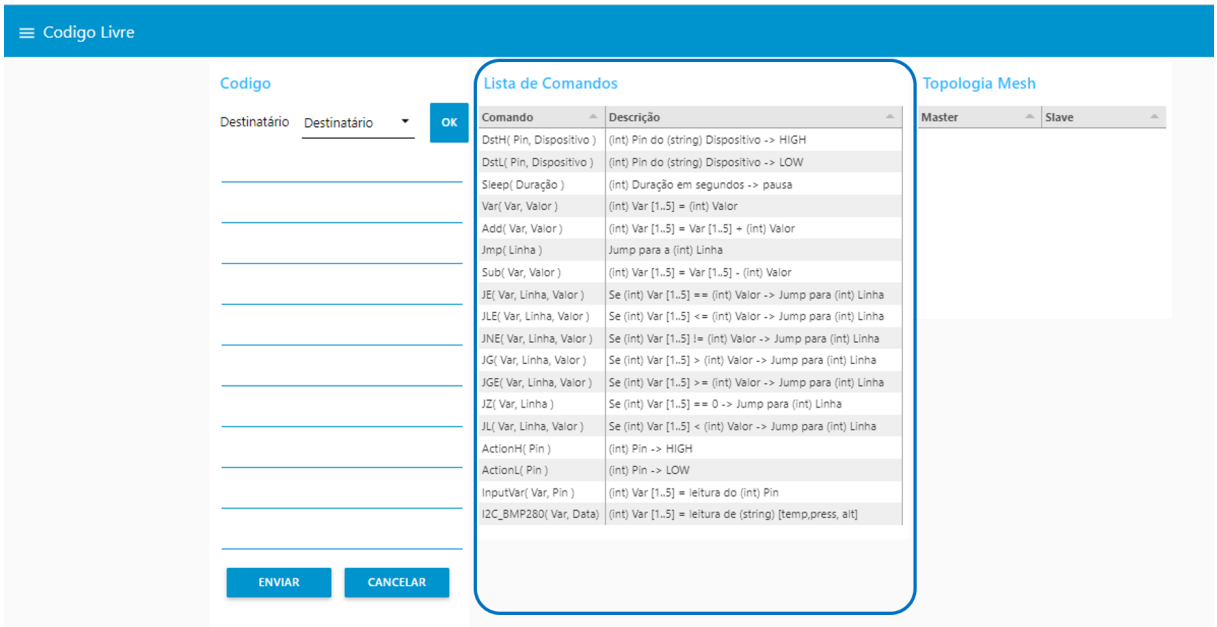


Figura 3.34: Lista de Comandos

A zona "Lista de Comandos"(figura 3.34) como o próprio nome indica é a lista de comandos disponíveis para implementar na rede *mesh*. O utilizador pode consultar a sintaxe aplicada a cada comando assim como uma breve descrição da sua funcionalidade.

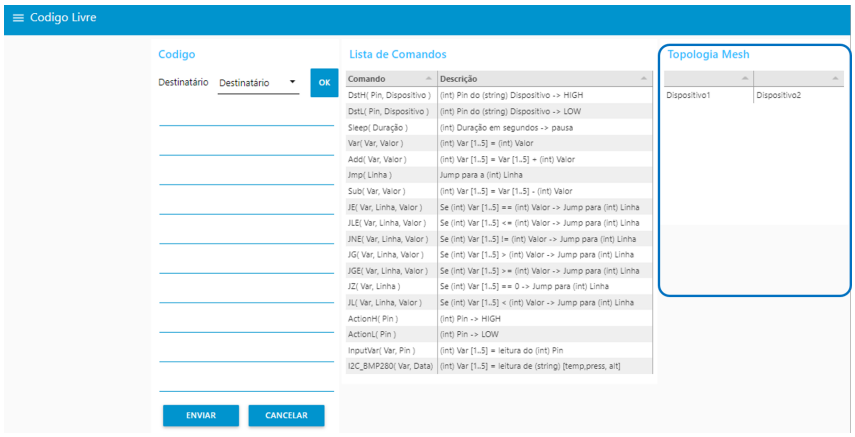


Figura 3.35: Topologia *Mesh*

A zona "Topologia *Mesh*"é uma representação das diversas ligações entre os dispositivos Secundários que foram implementadas pelo utilizador. Sempre que o utilizador implementa um comando que implique a interação entre dois dispositivos Secundário (DstH e DstL) essa relação fica registada, dando uma perspectiva, ao utilizador, das varias relações presentes na rede de dispositivos *mesh*.

3.3.5 Configurações *The Things Network*

Toda a solução desenvolvida, relativamente à utilização do protocolo LoRaWAN para estabelecer uma comunicação de longo alcance, está assente sobre a infraestrutura de rede TTN, por isso, neste capítulo serão abordadas todas as configurações e parametrizações necessárias para garantir a comunicação entre os diversos elementos.

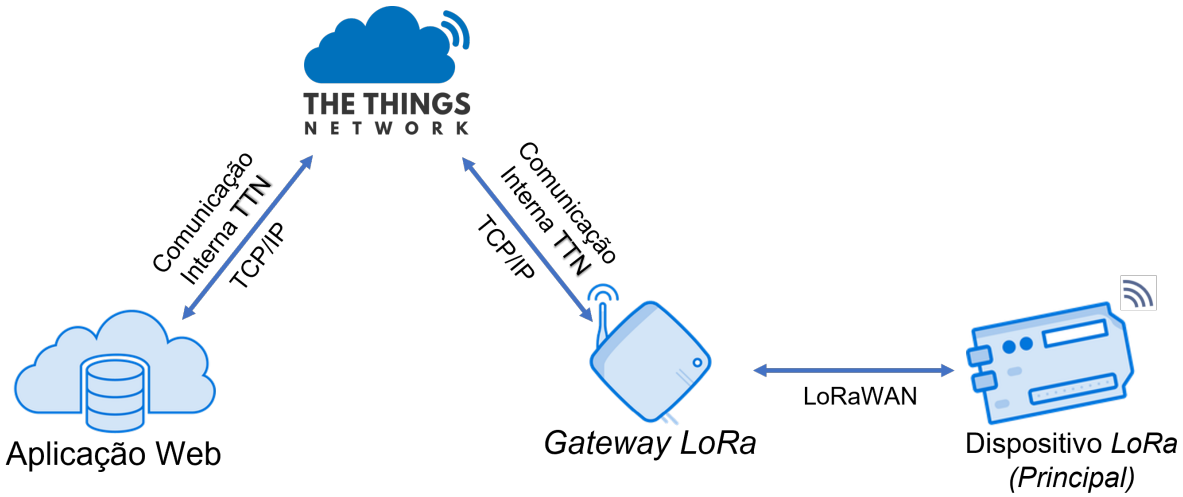


Figura 3.36: Arquitetura para comunicação de longo alcance

Na figura 3.36 está representada a arquitetura implementada neste projeto viabilizar

a interação da aplicação *Web* com a rede de dispositivos *Mesh*, sempre suportada pela plataforma TTN.

Para usufruir dos recursos e funcionalidades da infraestrutura de rede TTN é necessário criar uma conta de utilizador.

Com a conta criada, acedendo à área da consola, o primeiro passo é criar uma aplicação e nessa aplicação registar um dispositivo.

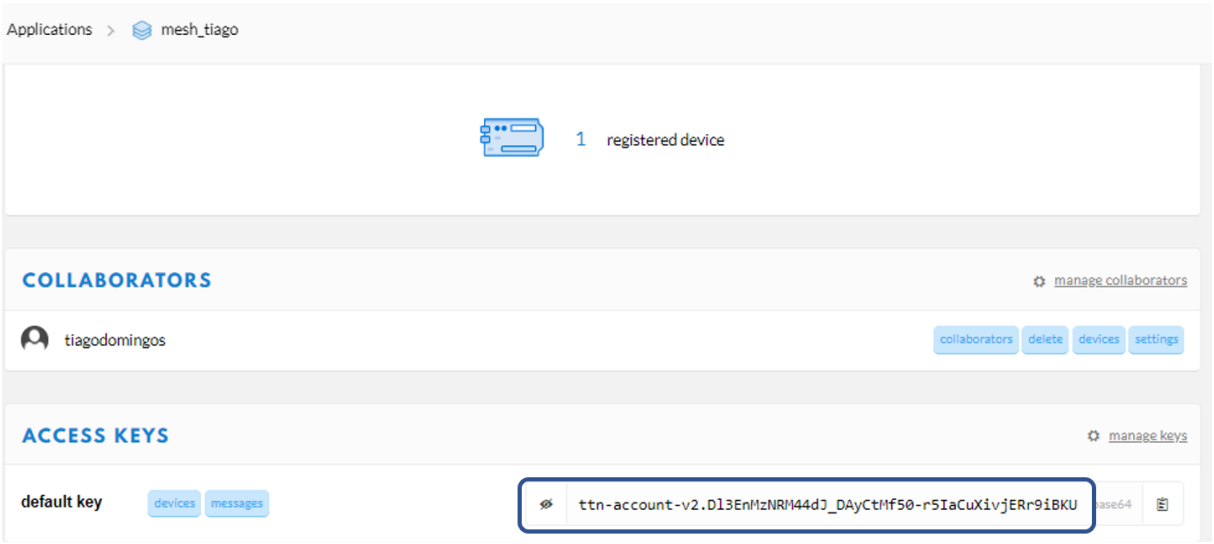


Figura 3.37: Credenciais da aplicação TTN

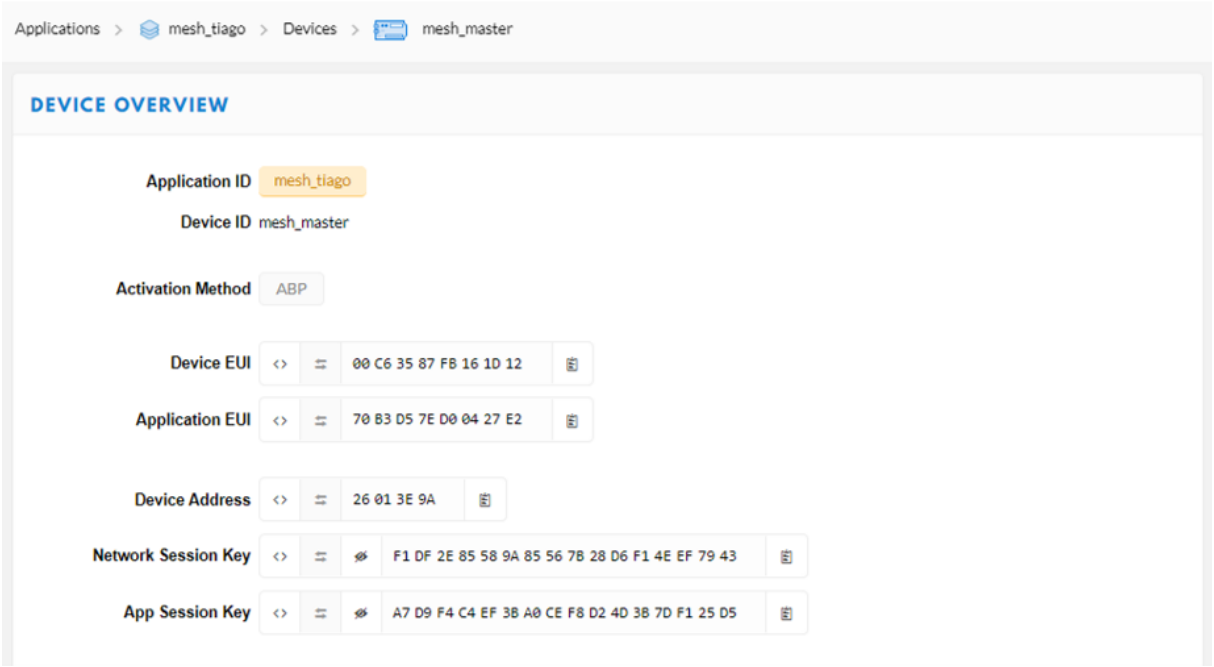


Figura 3.38: Credenciais do dispositivo TTN

Ao registar uma aplicação e o respectivo dispositivo o utilizador tem acesso a um conjunto de credenciais (figuras 3.37 e 3.38) e são estas credenciais que permitem a interface da infraestrutura de rede (TTN) com a aplicação de gestão e com o dispositivo Principal.

Para a aplicação de gestão e controlo (capítulo 3.3.4) tirou-se partido de uma biblioteca existente no *Node-RED* "*node-red-contrib-ttn*"para a comunicação com a TTN.

É a utilização desta biblioteca que permite o envio de mensagens através da TTN para a rede *mesh* (*downlink*) e a receção de mensagens através da TTN vindas da rede *mesh* (*uplink*).

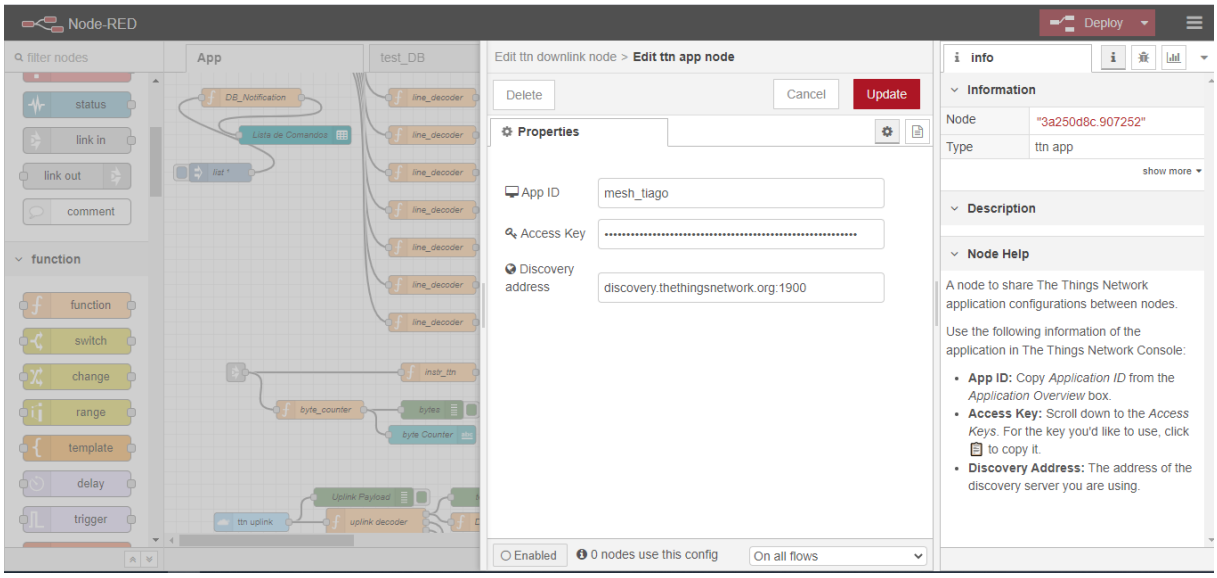


Figura 3.39: Configuração *Node-RED* para *dowlink* - parte 1

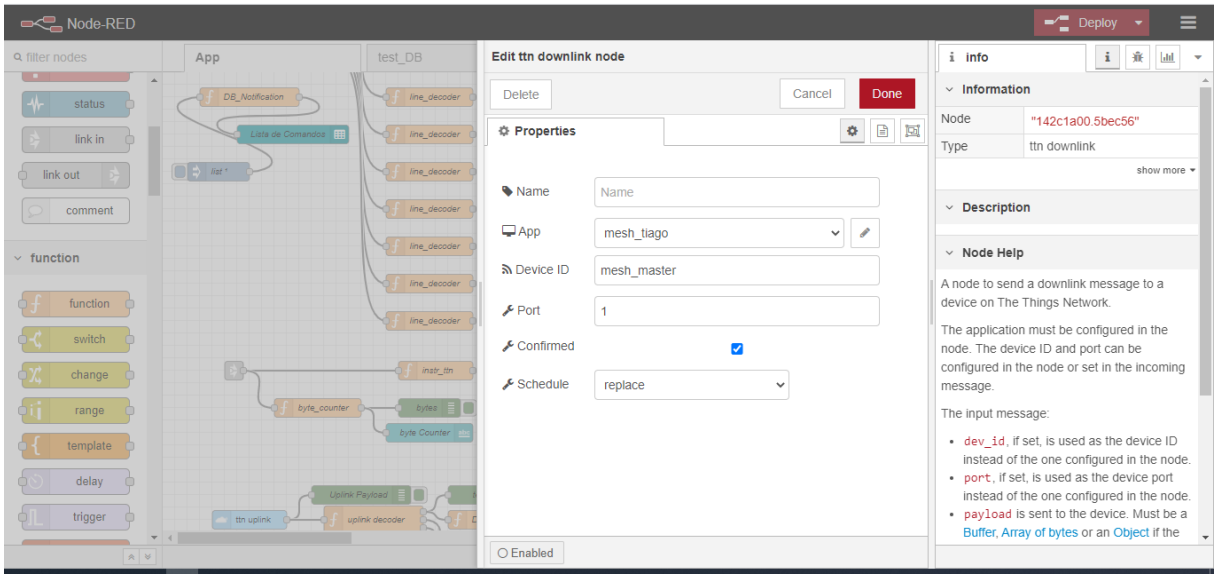


Figura 3.40: Configuração *Node-RED* para *dowlink* - parte 2

Nas figuras 3.39 e 3.40 estão presentes os parâmetros necessários para a configuração do fluxo responsável pelo o envio de informação em *downlink* para a rede *mesh*. Os parâmetros a configurar são o nome da aplicação TTN e a*access key* (figura 3.39) e o *Device ID* e porto (figura 3.40). As credenciais a utilizar são as geradas pela plataforma TTN (figuras 3.37 e 3.38).

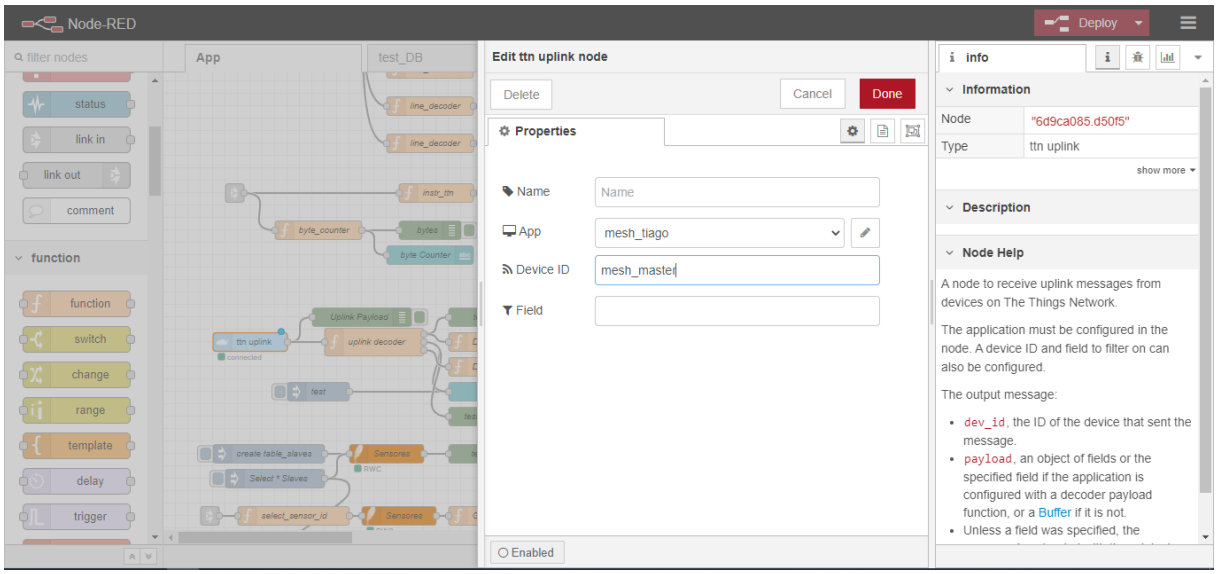


Figura 3.41: Configuração *Node-RED* para *uplink*

O fluxo desenvolvido para a recolha da informação vinda da rede *mesh* (*uplink*) é configurada de forma semelhante (figura 3.41) sendo apenas necessário o nome da aplicação TTN e o *Device ID*.

Para a configuração implementada no dispositivo Principal utilizou-se a biblioteca "LMIC"[24] disponível no ambiente de desenvolvimento *Arduino IDE*. A integração desta biblioteca permite a comunicação entre o dispositivo Principal e a rede LoRaWAN, neste caso a TTN.

Existem duas formas para o dispositivo Principal se conectar à TTN: *"Over the Air Activation"(OTAA)* e *"Activation by Personalization"(ABP)*.

O método de autenticação utilizado neste projeto foi o *ABP* sendo, por isso, necessário configurar o dispositivo Principal com os seguintes parâmetros:

- *Device Address (DevAddr)*;
- *Network Session Key (NwkSKey)*;
- *App Session Key (AppSKey)*.

Na figura 3.42 está representado um excerto do algoritmo desenvolvido para o dispositivo Principal onde são incluídos os parâmetros necessários para a comunicação com a rede: *Network Session Key*, *App Session Key* e *Device Address*. As credenciais a utilizar são as geradas pela plataforma TTN para o dispositivo registado (figura 3.38). Desta forma o dispositivo Principal consegue interagir com uma *Gateway LoRa* da TTN.

Figura 3.42: Configuração no Dispositivo Principal

Outro parâmetro que é necessário configurar para garantir uma comunicação o mais eficiente possível com uma *gateway LoRa* é o SF. Conforme explicado no capítulo 2.1.1 o protocolo LoRa/LoRaWAN utiliza seis SFs programáveis que vão desde SF7 a SF12.

Por definição o SF configurado pela biblioteca *LMIC* é um SF9, mas nos primeiros testes efectuados verificou-se que este valor de SF era insuficiente para o estabelecimento de uma comunicação estável. Para conseguir comunicar com uma *Gateway LoRa* e deste forma testar a plataforma de gestão desenvolvida, foi necessário aumentar o valor de SF9 para SF12.

A partir da consola disponibilizada TTN é possível analisar as mensagens trocadas entre o dispositivo Principal e a *gateway LoRa* e dos diversos parâmetros recolhidos um deles é o nome da *gateway LoRa*. Com esta informação e observando o mapa (figura 3.43) disponibilizado pela TTN é possível descobrir, geograficamente, a qual *gateway* o dispositivo se ligou.

Com informação da localização recolhida foi possível determinar que a distância entre a *Gateway* e o Principal era cerca de 15km. Sendo já uma distância considerável pode estar aqui o motivo para a necessidade da utilização de um SF12 em vez do SF9, uma vez que o aumento do SF resulta num maior alcance mas por outro lado também aumenta os recursos rádio utilizados e o atraso na comunicação, o que torna a utilização deste SF pouco viável.

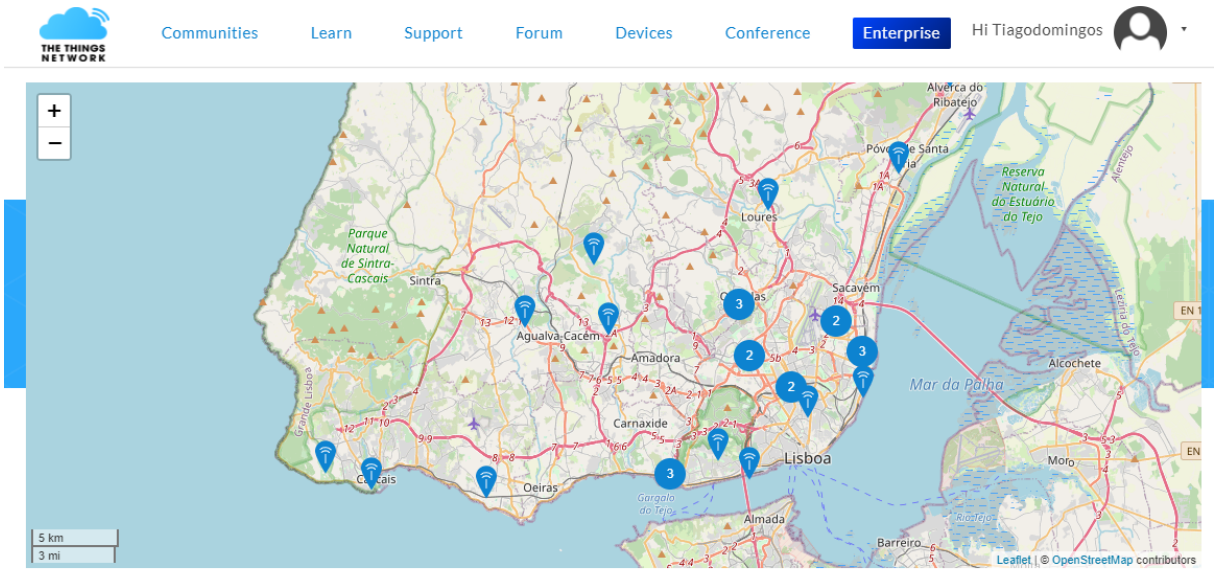


Figura 3.43: Mapa com a localização das *Gateway LoRa*

Posteriormente, com a instalação de uma *Gateway* mais próxima da localização do dispositivo Principal, tendo a distância reduzido para cerca de metade, já foi possível utilizar um SF9 e manter uma comunicação estável permitindo testar a plataforma de gestão e controlo da rede *Mesh* de forma mais eficaz. Este parâmetro tem de ser sempre ajustado conforme a localização do dispositivo Principal.

3.3.6 Configurações *ESP-Mesh*

Conforme apresentado no capítulo 3.2.1 o sistema é constituído por uma rede *Mesh* de dispositivos que utiliza o protocolo *ESP-Mesh* para comunicarem entre si.

Para implementar a rede *Mesh* tirou-se partido da biblioteca *"painlessMesh"* [25] disponível no ambiente *Arduino IDE*. Esta biblioteca possuí diversas funções de suporte à integração de uma rede *Wi-Fi*, mas o seu principal objectivo, como o próprio nome indica, é criar a rede *mesh* com o menor esforço possível para o programador. Ao implementar esta biblioteca tanto no dispositivo Principal como nos vários dispositivos Secundários basta que estes iniciem a sua operação para se tornarem automaticamente nós desta rede *Mesh*.

Entre as varias funções disponibilizadas pela *"painlessMesh"* as utilizadas para assegurar a comunicação entre os vários elementos da rede *Mesh* são as seguintes:

- *sendSingle(dest, msg)*: Função responsável pelo envio de uma mensagem para um determinado dispositivo presente na rede *Mesh*.

- *receivedCallback(from, msg)*: Sempre que um nó recebe uma mensagem, esta função é chamada. O campo *"from"* é o id do emissor da mensagem e o campo *"msg"* é a mensagem recebida.
- *changedConnectionCallback()*: Sempre que existe uma alteração na rede *Mesh*.

Todo o algoritmo desenvolvido é baseado na utilização das funções acima descritas, tendo sido implementadas tanto no dispositivo Principal como nos dispositivos Secundários.

A função *sendSingle* é utilizada em duas situações:

- Quando o dispositivo Principal necessita de enviar um programa para um dispositivo Secundário executar (tabela 3.3).
- Quando um dispositivo Secundário necessita de enviar um comando de ação para outro dispositivo Secundário (tabela 3.4).

A função *receivedCallback* permite a um nó receber uma mensagem proveniente de outro nó e atuar de acordo com a informação recebida sendo particularmente importante em dois cenários:

- Sempre que um dispositivo Secundário recebe um programa para executar proveniente do dispositivo Principal;
- Sempre que um dispositivo Secundário recebe um comando para executar proveniente de outro dispositivo Secundário ;

Associados a estas funções *Mesh* estão os algoritmos de codificação e descodificação descritos, respectivamente, nos capítulos 3.3.3.2 e 3.3.3.3. O primeiro antecede a utilização da função *sendSingle* e o segundo entra em execução depois da chamada da função *receivedCallback*.

A função *changedConnectionCallback* permite que o dispositivo Principal tenha uma visão geral sobre os vários elementos que compõem a rede *Mesh*, sempre que um novo elemento é adicionado ou removido esta função é chamada. Desta forma o dispositivo Principal consegue saber quais os dispositivos *online* e informar a aplicação de gestão com essa informação (figura 3.12).

A utilização desta biblioteca permite uma elevada abstração de todo o processo de gestão e controlo da rede *Mesh*, tirando partido das varias funções disponibilizadas para o desenvolvimento do algoritmo pretendido.

4

Conclusões

Neste capítulo é apresentado um sumário de todo o trabalho realizado durante a elaboração deste projeto. Também são enunciadas algumas propostas possíveis de trabalho futuro.

4.1 Trabalho Desenvolvido

O principal objetivo deste projeto era desenvolver uma plataforma para interface, controlo e gestão de uma rede de dispositivos, de simples interação para o comando dos vários elementos que habitualmente compõem as ações performativas realizadas pela Musa Paradisíaca.

Antes de iniciar o processo de desenvolvimento e implementação da plataforma foi necessário, atendendo aos requisitos de projeto, definir os seguintes tópicos:

- Tecnologia LPWAN a utilizar para a comunicação entre a aplicação *web* e a rede de dispositivos;
- Ferramenta de programação para o desenvolvimento da aplicação para interface com o utilizador;
- Micro-controlador integrado nos dispositivos;
- Protocolo de comunicação a utilizar pela rede de dispositivos.

Com estes tópicos esclarecidos definiu-se a arquitetura física e lógica do sistema a implementar.

Uma vez definida a arquitetura de todos os componentes do sistema procedeu-se à elaboração e desenvolvimento de uma linguagem onde se definiu o formato de toda a informação (útil) transmitida entre os vários elementos do sistema e que permita ao utilizador, a partir da aplicação, instruir a rede de dispositivos.

Associado ao desenvolvimento desta linguagem esteve também a implementação de um algoritmo de codificação e descodificação (implementado nos dispositivos e na aplicação *web*) de todas as mensagens trocadas entre a aplicação *web* e a rede *Mesh* (e vice-versa) quer seja o envio e receção de comandos ou outro tipo de informação transmitida.

Com a linguagem e os processos inerentes à mesma definidos, procedeu-se ao *design* da aplicação de gestão e controlo da rede *Mesh*, onde foram definidas duas áreas: uma de gestão da rede e dos dispositivos e outra onde o utilizador tem a possibilidade de enviar instruções para os vários dispositivos de rede.

Foi também necessário realizar alguns processos de configuração relacionados com o protocolo *ESP-Mesh* e com a infraestrutura de rede IoT TTN para garantir o correto interface entre os vários elementos.

4.2 Trabalho Futuro

Como trabalho futuro, existem alguns aspetos que podem ser melhorados, otimizando o trabalho desenvolvido. Esses aspetos são:

- Aumentar o número de comandos reconhecidos pelos dispositivos, diversificando assim os cenários possíveis de implementar pelo utilizador.
- Implementar comandos para parametrização das configurações associadas à TTN e ao protocolo *ESP-Mesh*, reduzindo assim a necessidade de ligação física ao dispositivo para o configurar.
- Otimizar a codificação das mensagens, reduzindo assim o tamanho de cada mensagem e consequentemente o recurso rádio utilizado.
- Apresentar mais informação sobre a rede de dispositivos na aplicação de gestão e controlo.
- Desenvolvimento de uma versão da aplicação para (por exemplo) *Android*, oferecendo ao utilizador maior mobilidade para interagir com a aplicação.

Referências

[1] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of lpwan technologies for large-scale iot deployment," ICT express, vol. 5, no. 1, pp. 1–7, 2019.

[2] M. El-Aasser, A. Gasser, M. Ashour, and T. Elshabrawy, "Performance analysis comparison between lora and frequency hopping-based lpwan," in 2019 IEEE Global Conference on Internet of Things (GCIoT), pp. 1–6, IEEE, 2019.

[3] "*The Things Network*." `https://www.thethingsnetwork.org/`. Accessed: Abril 2021.

[4] "*The Things Indoor Gateway*." `https://www.thethingsnetwork.org/docs/gateways/thethingsindoor/`. Accessed: Abril 2021.

[5] "*The Things Gateway*." `https://www.thethingsnetwork.org/docs/gateways/gateway/`. Accessed: Abril 2021.

[6] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios," IEEE Wireless Communications, vol. 23, no. 5, pp. 60–67, 2016.

[7] L. Vangelista, A. Zanella, and M. Zorzi, "Long-range iot technologies: The dawn of lora™," in Future access enablers of ubiquitous and intelligent infrastructures, pp. 51–58, Springer, 2015.

[8] A. Lavric, A. I. Petrariu, and V. Popa, "Long range sigfox communication protocol scalability analysis under large-scale, high-density conditions," IEEE Access, vol. 7, pp. 35816–35825, 2019.

[9] M. Lauridsen, I. Z. Kovács, P. Mogensen, M. Sorensen, and S. Holst, "Coverage and capacity analysis of lte-m and nb-iot in a rural area," in 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall), pp. 1–5, IEEE, 2016.

[10] W. Ayoub, M. Mroue, F. Nouvel, A. E. Samhat, and J.-C. Prévotet, "Towards ip over lpwans technologies: Lorawan, dash7, nb-iot," in 2018 sixth international conference on digital information, networking, and wireless communications (dinwc), pp. 43–47, IEEE, 2018.

[11] Y.-P. E. Wang, X. Lin, A. Adhikary, A. Grovlen, Y. Sui, Y. Blankenship, J. Bergman, and H. S. Razaghi, "A primer on 3gpp narrowband internet of things," IEEE communications magazine, vol. 55, no. 3, pp. 117–123, 2017.

[12] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "Overview of cellular lpwan technologies for iot deployment: Sigfox, lorawan, and nb-iot," in 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 197–202, IEEE, 2018.

[13] Arduino, "Arduino uno wifi rev2." https://store.arduino.cc/arduino-uno-wifi-rev2, 2020. Acedido: Abril 2021.

[14] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things," in 2017 Internet Technologies and Applications (ITA), pp. 143–148, IEEE, 2017.

[15] E. Systems, "Esp32 technical reference manual." https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf, 2021. Acedido: Fevereiro 2021.

[16] E. Systems, "Esp-mesh - api guide." https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides, 2020. Acedido: Abril 2020.

[17] K. Mikhaylov and J. Tervonen, "Multihop data transfer service for bluetooth low energy," in 2013 13th international Conference on ITS Telecommunications (ITST), pp. 319–324, IEEE, 2013.

[18] R. Gupta and R. Gupta, "Abc of internet of things: Advancements, benefits, challenges, enablers and facilities of iot," in 2016 Symposium on Colossal Data Analysis and Networking (CDAN), pp. 1–5, IEEE, 2016.

[19] M. Jürgens, D. Meis, D. Möllers, F. Nolte, E. Stork, G. Vossen, C. Werner, and H. Winkelmann, "Bluetooth mesh networks for indoor localization," in 2019 20th IEEE International Conference on Mobile Data Management (MDM), pp. 397–402, IEEE, 2019.

[20] E. Systems, "Esp-ble-mesh - esp idf programming guide." https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-index.html, 2020. Acedido: Abril 2020.

[21] M. F. Coniglio, "Isadora - user manual." https://troikatronix.com/files/isadora-manual.pdf, 2019. Acedido: Abril 2021.

[22] "support.troikatronix.com." https://support.troikatronix.com/support/solutions/articles/13000056359-pricing-for-isadora-3. Acedido: Maio 2021.

[23] "cycling74." https://cycling74.com/shop. Acedido: Maio 2021.

[24] M. Kooijman, "Arduino-lmic library." https://github.com/matthijskooijman/arduino-lmic, 2020. Acedido: Maio 2020.

[25] "Arduino - painlessmesh library." https://gitlab.com/painlessMesh/painlessMesh, 2020. Acedido: Maio 2020.