instituto superior de engenharia de lisboa

Relatório Final do Projeto Final de Curso

Sistema de Rega Inteligente com Controlo Local e Comunicação LoRa

Afonso Ferreira, 49340
A49340@alunos.isel.pt

Orientador: Pedro Sampaio
pedro.sampaio@isel.pt

Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Eletrónica e Telecomunicações e Computadores

2.º Semestre 2025/2026

Resumo

O presente relatório documenta o desenvolvimento de um protótipo funcional de sistema de rega inteligente para jardins de pequena e média dimensão, baseado num concentrador ESP32-S3 e em controladores de válvula STM32L432KC e LPC1769 com comunicação Long Range (LoRa) ponto-a-ponto.

A solução implementada inclui controlo local por interface Web e API HTTP, persistência local em LittleFS/Non-Volatile Storage (NVS), protocolo binário compacto para LoRa, serviço rádio com módulos RA-02/SX1278 e firmware FreeRTOS para os controladores de válvula. O modelo operacional adotado associa cada controlador a uma válvula e mantém, no concentrador, no máximo um comando pendente por controlador.

O sistema suporta emparelhamento dos controladores, envio periódico de estado, entrega de comandos de abertura ou fecho e confirmação de execução através de `COMMAND_ACK`. Para reduzir colisões repetidas, foram usados atrasos aleatórios limitados em novas tentativas e verificações periódicas, sem introduzir uma camada de acesso ao meio mais complexa.

A validação realizada confirma a comunicação LoRa real entre o concentrador e os dois controladores, bem como o funcionamento integrado da interface Web, da API HTTP, do protocolo, da persistência e do ciclo de comando. Nos controladores, a atuação foi representada pelos LEDs integrados das placas; não foram integrados o driver de potência nem a válvula hidráulica. O resultado é um protótipo operacional, ainda com limitações ao nível da escala, segurança, autonomia energética e preparação para instalação final.

Palavras-chave: IoT, LoRa, rega inteligente, ESP32-S3, controlo local.

Abstract

This report documents the development of a working smart irrigation prototype for small and medium-sized gardens, based on an ESP32-S3 concentrator and STM32L432KC/LPC1769 valve controllers using point-to-point LoRa communication.

The implemented solution includes local control through a Web interface and HTTP API, local LittleFS/NVS persistence, a compact binary LoRa protocol, an RA-02/SX1278 radio service, and FreeRTOS firmware for the valve controllers. The operating model maps each controller to one valve and keeps at most one pending command per controller in the concentrator.

The system supports controller pairing, periodic status reporting, delivery of open/close commands, and execution confirmation through `COMMAND_ACK`. Bounded random delays are used in retries and periodic check-ins to reduce repeated collisions without introducing a more complex medium-access layer.

Validation confirms real LoRa communication between the concentrator and the two controllers, as well as the integrated operation of the Web interface, HTTP API, protocol, persistence, and command cycle. Valve actuation was represented by the boards' on-board LEDs; neither the power driver nor a hydraulic valve was integrated. The result is an operational prototype, with remaining limitations in scale, security, battery autonomy, and readiness for final installation.

Keywords: IoT, LoRa, smart irrigation, ESP32-S3, local control.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ACK Acknowledgment

AP Access Point

API Interface de Programação de Aplicações

BLE Bluetooth Low Energy

CAD Channel Activity Detection

CRC Cyclic Redundancy Check

CSS Cascading Style Sheets

DNS Domain Name System

ESP-IDF Espressif IoT Development Framework

FreeRTOS Free Real-Time Operating System

GPIO General-Purpose Input/Output

HTML HyperText Markup Language

HTTP Protocolo de Transferência de Hipertexto

IEEE Institute of Electrical and Electronics Engineers

IoT Internet das Coisas

IRQ Interrupt Request

JSON JavaScript Object Notation

LoRa Long Range

LoRaWAN Long Range Wide Area Network

MCU Microcontroller Unit

mDNS Multicast Domain Name System

NVS Non-Volatile Storage

P2P Peer-to-Peer

RSSI Received Signal Strength Indicator

RTC Real-Time Clock

RX Receive

SNR Signal-to-Noise Ratio

SNTP Simple Network Time Protocol

SoC System on Chip

SPI Serial Peripheral Interface

SSP Synchronous Serial Port

STA Station

TCP/IP Transmission Control Protocol / Internet Protocol

TDMA Time Division Multiple Access

TX Transmit

WFI Wait For Interrupt

Glossário

O presente glossário reúne um conjunto reduzido de termos técnicos utilizados ao longo do relatório, com o objetivo de assegurar uma leitura homogénea.

Hub-and-spoke Topologia em estrela na qual um dispositivo central (*hub*) coordena a comunicação com múltiplos dispositivos periféricos (*spokes*), sem comunicação direta entre estes.

Runtime Período durante o qual o software se encontra em execução. No relatório, os perfis de *runtime* representam a distribuição das amostras pelas funções observadas durante esse período.

Check-in Mensagem periódica enviada pelo controlador para reportar a sua presença e o seu estado ao concentrador, servindo simultaneamente de janela de oportunidade para a receção de comandos.

Peer-to-peer Modo de comunicação direto entre dois dispositivos, sem recurso a infraestrutura intermediária, nomeadamente *gateways* ou servidores de rede.

Painel principal (*dashboard*) Página inicial da interface Web que apresenta uma visão agregada do estado do sistema e o acesso às principais operações sobre os controladores.

Detalhe do controlador Página da interface Web dedicada a um controlador individual, onde são apresentados a fila ativa de comandos, o histórico e os parâmetros configurados.

Callback Função registada num módulo para ser invocada quando ocorre um evento ou quando é necessária uma operação fornecida por outra camada.

Flame graph Representação de amostras da pilha de chamadas em que cada *frame* corresponde a uma função e a sua largura é proporcional ao número de amostras observadas.

Gateway Elemento de uma rede LoRaWAN que encaminha o tráfego rádio dos dispositivos para o servidor de rede através de uma ligação IP.

Idle hook Função invocada pela tarefa inativa do FreeRTOS quando não existem outras tarefas prontas a executar.

Namespace Espaço lógico que agrupa e separa chaves numa memória persistente organizada por pares chave–valor.

Pinout Correspondência entre os pinos físicos de um componente e as respetivas funções ou ligações elétricas.

Standby Estado de espera em que o dispositivo preserva o contexto necessário a um retomar rápido, com menor atividade do que no modo ativo.

Timeout Limite máximo de espera após o qual uma operação é considerada expirada ou sem resposta.

Nonce Valor de utilização única que permite distinguir uma troca de mensagens e relacionar

um pedido com a respetiva resposta.

Jitter Variação aleatória introduzida num intervalo temporal para reduzir a probabilidade de transmissões simultâneas.

Backoff Período de espera aplicado antes de uma nova tentativa de comunicação, com o objetivo de reduzir colisões e utilização excessiva do canal.

Duty cycle Fração do tempo total durante a qual um dispositivo ou componente permanece ativo, normalmente expressa em percentagem.

Solenoide biestável
Atuador que mantém uma de duas posições estáveis sem alimentação contínua e muda de posição através de um impulso elétrico.

RSSI Indicador da potência do sinal recebido, normalmente expresso em dBm; valores menos negativos representam, em geral, um sinal mais forte.

SNR Relação entre a potência do sinal e a potência do ruído, expressa em dB; valores superiores indicam, em geral, melhores condições de comunicação.

1 Introdução

1.1 Contexto do Projeto

O presente projeto consiste no desenvolvimento de um sistema de rega inteligente, direcionado a jardins de pequena e média dimensão. Neste relatório, a expressão rega inteligente designa a gestão programável e local da rega, incluindo configuração de horários, envio de comandos e monitorização do estado. O protótipo não toma decisões autónomas com base em sensores ou em dados meteorológicos. A solução proposta visa minimizar a complexidade de instalação e a necessidade de manutenção, assegurando simultaneamente eficiência energética e operação com controladores de válvula alimentados a bateria.

O sistema baseia-se numa arquitetura distribuída, composta por um concentrador local e por controladores associados às válvulas de rega, permitindo o controlo local do sistema e a configuração de horários de rega. A comunicação entre os diferentes elementos recorre a uma tecnologia sem fios de baixo consumo.

No protótipo implementado, esta arquitetura materializa-se num concentrador ESP32-S3, em controladores FreeRTOS STM32L432KC e LPC1769, comunicação LoRa ponto-a-ponto a 433 MHz com módulos RA-02/SX1278, uma interface Web local e uma API HTTP. Cada controlador representa uma válvula, mantendo simples o modelo de controlo e validação. No hardware validado, a atuação foi simulada pelos LEDs integrados das placas; não foi ligada uma válvula hidráulica.

A arquitetura foi concebida de forma modular para permitir a adição de controladores. Contudo, a validação experimental foi realizada apenas com dois controladores, pelo que o comportamento do sistema com um número superior de dispositivos não foi testado.

1.2 Motivação

A motivação para o desenvolvimento deste projeto surge da necessidade de automatizar sistemas de rega em contexto residencial, onde as soluções existentes apresentam limitações ao nível do custo, da complexidade de instalação e da manutenção.

Sistemas convencionais requerem frequentemente cablagem dedicada entre o controlador central e as válvulas de rega, implicando intervenções físicas no terreno e aumentando significativamente o custo e a complexidade da instalação, sobretudo em infraestruturas já existentes.

Neste contexto, pretende-se desenvolver uma solução alternativa que:

- elimine a necessidade de cablagem dedicada;
- reduza os custos de instalação;
- assegure baixo consumo energético;
- permita uma utilização simples e intuitiva por parte do utilizador final.

1.3 Estrutura do Relatório

O relatório encontra-se organizado de forma progressiva. Inicialmente, são apresentados o enquadramento do problema e a definição dos objetivos do projeto. Em seguida, é realizada a análise do estado

da arte e das tecnologias relevantes.

Posteriormente, são descritas a arquitetura do sistema e a implementação do protótipo. Por fim, são apresentados os resultados de validação do protótipo implementado, as limitações atuais e as possíveis evoluções futuras do sistema.

1.4 Disponibilização Web do Projeto

Foi criada uma página Web pública de apresentação do projeto, disponível em:

`https://wireless-irrigation-control.me/`

Esta página serve como ponto de acesso externo para a divulgação do sistema desenvolvido e para a consulta de informação complementar sobre o protótipo.

O repositório de código-fonte do projeto está disponibilizado em `https://github.com/sucumbap/wireless-irrigation`.

1.5 Nota sobre Ferramentas de Apoio

Importa ainda referir que, durante a elaboração deste relatório, foi utilizada uma ferramenta de Inteligência Artificial generativa da OpenAI, como apoio à revisão linguística, estilística e estrutural do texto.

2 Enquadramento, Objetivos e Estado da Arte

2.1 Problema a Resolver

O objetivo do projeto consiste no desenvolvimento de um sistema de rega inteligente que permita o controlo local e a configuração de horários de rega de forma eficiente, com reduzida complexidade de instalação e menor dependência de cablagem dedicada entre o controlador e as válvulas.

Nos sistemas convencionais, a ligação física entre controlador e eletroválvulas pode aumentar custos de instalação e dificultar alterações futuras. Neste contexto, a solução proposta explora uma arquitetura com concentrador local e controladores de cada válvula com ligação rádio, mantendo as operações essenciais disponíveis localmente.

O sistema deve, assim:

- suportar comunicação sem fios entre o concentrador e os controladores de cada válvula;
- permitir instalação em ambientes residenciais com intervenção reduzida na infraestrutura existente;
- disponibilizar controlo e configuração por meio de uma interface Web local;
- considerar eficiência energética nos controladores de cada válvula, pois estes podem ser alimentados a bateria.

2.2 Objetivos e Âmbito do Protótipo

Os objetivos do protótipo combinam a validação funcional do sistema de rega com a implementação dos blocos necessários ao seu funcionamento local. Assim, pretende-se:

- desenvolver um protótipo funcional de um sistema de rega inteligente, orientado para utilização residencial;
- implementar um concentrador local responsável pela configuração de horários, pela gestão dos controladores de válvula e pela disponibilização da interface Web e da API HTTP;
- implementar controladores capazes de receber comandos, atualizar uma saída digital que simula a atuação da válvula e confirmar a execução;
- definir um protocolo leve para emparelhamento, envio de estado, entrega de comandos e confirmação de execução;
- integrar comunicação LoRa ponto-a-ponto entre o concentrador e os controladores de válvula;
- manter persistência local das configurações necessárias ao funcionamento autónomo do sistema;
- validar o ciclo essencial de operação com hardware real de processamento e comunicação: emparelhamento, envio de estado, entrega de comando, atualização da saída de atuação simulada e confirmação de execução. O driver de potência e a válvula hidráulica real não foram integrados no protótipo.

No âmbito implementado, o protótipo inclui o concentrador ESP32-S3, controladores físicos STM32L432KC e LPC1769, comunicação LoRa ponto-a-ponto sobre RA-02/SX1278, interface Web local, API HTTP e

persistência local em NVS e LittleFS. O concentrador mantém os horários, o registo de controladores, os comandos em espera e o estado durável da aplicação; cada controlador atualiza o estado lógico representado pelo LED integrado da placa, envia estados e confirma a execução de comandos recebidos.

A validação experimental foi realizada com dois controladores físicos FreeRTOS. A arquitetura e o modelo de dados do concentrador suportam múltiplos controladores, mas a caracterização de escala com uma rede extensa não é assumida como concluída nesta fase.

2.3 Objetivo da Análise

Neste capítulo analisam-se soluções de rega inteligente e tecnologias de comunicação sem fios relevantes. A análise não pretende repetir os objetivos definidos anteriormente, mas identificar alternativas e limitações que influenciam a arquitetura adotada.

Dado o caráter aplicado do projeto, foram consideradas referências académicas, documentação técnica e soluções comerciais com relevância para uma instalação residencial baseada em infraestrutura hidráulica existente e operação local.

2.4 Soluções de Rega Existentes

2.4.1 Produtos Comerciais

No contexto comercial, identificam-se três abordagens principais.

A primeira é representada por soluções com controlador central, exemplificadas por plataformas como Rain Bird, Hydrawise e Rachio [1, 2, 3]. Neste tipo de solução, a configuração e a lógica de controlo tendem a concentrar-se num controlador dedicado, frequentemente integrado num ecossistema próprio de gestão da rega.

A segunda abordagem é exemplificada por dispositivos autónomos de instalação local, tipicamente posicionados ao nível da torneira ou válvula [4, 5]. Estes dispositivos tendem a simplificar a instalação e a reduzir o custo inicial, mas são habitualmente mais adequados a cenários simples do que a sistemas com múltiplos pontos de rega distribuídos.

Importa ainda considerar soluções comerciais de automação modular, como o Shelly Pro LoRa Add-on, que adicionam comunicação LoRa a dispositivos compatíveis da gama Shelly Pro e são apresentadas para cenários de automação remota, incluindo aplicações agrícolas [6]. Embora relevantes pela utilização de comunicação de longo alcance, estas soluções não constituem, por si só, controladores de rega completos, dependendo do ecossistema Shelly e de configuração adicional.

No âmbito deste projeto, não se pretende substituir a infraestrutura hidráulica existente, mas sim introduzir uma camada de controlo sem fios, permitindo a modernização de sistemas já instalados com impacto mínimo na instalação.

2.4.2 Projetos Open-Source e Literatura Académica

No domínio open-source, soluções como o OpenSprinkler evidenciam a viabilidade técnica de sistemas configuráveis e flexíveis. No entanto, estas soluções exigem, em geral, maior envolvimento técnico nas fases de instalação, integração e manutenção [7].

Em contraste, a literatura académica recente sobre rega inteligente tende a privilegiar abordagens assentes em sensores, modelos de decisão e estratégias de otimização, recorrendo, por exemplo, a medições de humidade do solo e a variáveis meteorológicas para apoiar a gestão da rega [8].

O presente projeto segue uma abordagem distinta, centrada em:

- controlo direto pelo utilizador;
- operação previsível e configurável;

- simplicidade de utilização.

Esta abordagem permite reduzir a complexidade do sistema, privilegiando uma solução centrada no utilizador final e alinhada com os requisitos definidos para o projeto.

A Tabela 2.1 sintetiza as principais abordagens identificadas e a sua relevância para este projeto.

Tabela 2.1: Síntese das abordagens de soluções de rega existentes

Categoria	**Exemplo**	**Contributo principal**	**Limitação para este projeto**
Comercial (controlador central)	Rain Bird [1], Hydrawise [2], Rachio [3]	Ecossistema maduro e funcionalidades completas de gestão de rega.	Foco frequente em arquiteturas com cablagem/infraestrutura própria e maior dependência de plataforma.
Comercial (temporizador de torneira)	Orbit B-hyve [4], produtos tipo *smart timer* [5]	Entrada rápida e custo inicial reduzido.	Foco em zonas isoladas; menor adequação a sistemas com vários pontos de rega distribuídos.
Comercial (automação modular)	Shelly Pro LoRa Add-on [6]	Integra comunicação LoRa de longo alcance em dispositivos Shelly Pro, relevante para automação remota e cenários agrícolas.	Não constitui uma solução de rega completa; depende do ecossistema Shelly, de dispositivos compatíveis e de configuração adicional.
Open-source	OpenSprinkler [7]	Flexibilidade e controlo técnico elevado.	Exige maior esforço de integração e manutenção contínua.
Académico	Sistemas IoT com sensores de humidade e clima [8]	Otimização da rega baseada em dados.	Nem sempre prioriza instalação simples e controlo manual previsível do utilizador.

Com base na análise apresentada, verifica-se que as soluções existentes apresentam limitações ao nível da complexidade, dependência de infraestrutura ou foco excessivo em automação, justificando a abordagem adotada neste projeto.

2.5 Tecnologias de Comunicação Relevantes

Foram analisadas diversas tecnologias de comunicação sem fios, incluindo Wi-Fi, Zigbee/Thread (baseadas na norma IEEE 802.15.4 [9, 10]), LoRa em modo P2P [11], LoRaWAN [11, 12] e Bluetooth Low Energy (BLE) Mesh.

A seleção foi orientada pelos seguintes requisitos:

- baixo consumo energético;
- cobertura em ambiente exterior;
- suporte a comunicação entre múltiplos controladores distribuídos;
- independência de infraestrutura adicional.

As tecnologias de comunicação de curto alcance, incluindo as baseadas em IEEE 802.15.4 e Bluetooth, podem apresentar limitações de cobertura por ligação e de penetração em paredes e outros obstáculos [11].

A Tabela 2.2 apresenta uma síntese comparativa das tecnologias analisadas. Os alcances correspondem a valores máximos de referência e não a distâncias garantidas de operação [13].

Tabela 2.2: Comparação das tecnologias de comunicação analisadas

Tecnologia	**Banda**	**Topologia**	**Alcance indicativo**	**Adequação ao projeto**
Wi-Fi	2,4 GHz	Estrela (AP-cliente)	Até 100 m (por ligação)	Adequado ao concentrador; consumo elevado.
Zigbee/Thread (802.15.4)	2,4 GHz	Malha/estrela	Até 100 m (por ligação)	Eficiente, mas acrescenta complexidade de configuração ao protótipo atual.
LoRa P2P	Sub-GHz	Estrela com ligações diretas concentrador–controlador	Até 5 km (urbano) 15–20 km (rural)	Adequado ao protótipo: longo alcance, baixo consumo e sem *gateway*.
LoRaWAN	Sub-GHz	Estrela com *gateway*	Até 5 km (urbano) 15–20 km (rural)	Adequado a redes extensas, mas implica *gateway* e maior complexidade do que a necessária no protótipo.
BLE Mesh	2,4 GHz	Malha	Até 240 m (por ligação BLE)	Adequado a curtas distâncias; menos indicado no exterior.

O alcance real depende da potência de emissão, da sensibilidade do recetor, das antenas, dos obstáculos e dos parâmetros da camada física. Para Zigbee, a documentação oficial indica 10–100 m, consoante a potência de transmissão e as características do ambiente; Thread utiliza igualmente IEEE 802.15.4 e amplia a cobertura através da malha [14, 15]. Para LoRa, a documentação oficial aponta cerca de 5 km em ambiente urbano e até 15 km em ambiente rural, enquanto a comparação publicada na IEEE Access indica até 20 km em ambiente rural [16, 17, 13]. Os mesmos valores são apresentados para LoRa P2P e LoRaWAN porque ambos podem utilizar a mesma camada física. Por fim, o Bluetooth SIG não estabelece uma distância única para BLE; numa rede Mesh, a cobertura total depende também da quantidade e da disposição dos relés [18, 19].

Com base na análise apresentada, conclui-se que a tecnologia **LoRa** em modo ***peer-to-peer* (P2P)** apresenta o melhor compromisso entre alcance, consumo energético e complexidade de integração, sendo, por esse motivo, selecionada.

2.6 Plataformas de Microcontrolador para os Controladores de Válvula

A seleção da plataforma de microcontrolador para os controladores de válvula teve em consideração os seguintes critérios:

- baixo consumo energético;
- suporte a comunicação LoRa;
- simplicidade de integração;
- adequação à prototipagem.

Foram consideradas duas abordagens:

- utilização de um microcontrolador (Microcontroller Unit (MCU)) com rádio LoRa externo;
- utilização de System on Chip (SoC) com rádio LoRa integrado.

A Tabela 2.3 apresenta uma comparação das plataformas consideradas e dos aspetos mais relevantes para o projeto.

Tabela 2.3: Plataformas analisadas para os controladores de válvula

Tipo de plataforma	**Exemplos**	**Pontos fortes**	**Limitações no protótipo funcional atual**
SoC com rádio integrado	STM32WLE5 STM32WL55 [20]	Integra MCU e rádio sub-GHz no mesmo circuito, permitindo hardware final compacto.	Exige o desenvolvimento de uma placa e a integração direta do circuito de radiofrequência.
Módulo com SoC e rádio	Wio-E5 e RAK3172 [21, 22]	Reduz o esforço de integração do rádio e permite uma montagem compacta.	Oferece menor flexibilidade de hardware do que a utilização direta do SoC.
Placa de desenvolvimento ou avaliação	NUCLEO-WL55JC, LoRa-E5 Mini/Dev Kit, Grove LoRa-E5 e RAK3272S [23, 24, 25, 26, 27]	Facilita a programação e a validação funcional em bancada.	Tem maiores dimensões e não se destina à instalação final junto da válvula.
MCU + rádio externo (alternativa analisada)	ESP32-S3 + LR1121 [28]	Grande flexibilidade para iterações com Wi-Fi/BLE.	Maior complexidade energética e de integração face a um SoC LoRa dedicado.
MCU + rádio externo (protótipo validado)	STM32L432KC e LPC1769 com RA-02/SX1278 [29, 30, 31]	Demonstra a portabilidade do software entre plataformas FreeRTOS e a separação entre lógica comum e adaptação por placa.	Montagem menos compacta e menos otimizada do que uma placa final dedicada.

A Tabela 2.4 complementa a comparação qualitativa com valores de referência para plataformas representativas.

Tabela 2.4: Comparação quantitativa de plataformas representativas

Plataforma	**Memória**	**Dimensão**	**Consumo**	**Preço (USD) e disponibilidade**
Wio-E5 [21, 32]	Flash: 256 KB RAM: 64 KB	12 × 12 × 2,5	Sono: 2,1 μA RX: 6,7 mA TX: 111 mA a 22 dBm	\$6,99 Seeed Studio Em stock
RAK3172 [22, 33]	Flash: 256 KB RAM: 64 KB	15,5 × 15,5 × 2,6	Sono: 1,69 μA RX: 5,22 mA TX: 87 mA a 20 dBm	\$5,99–\$6,99 RAKwireless Disponível
NUCLEO-WL55JC [23, 34]	Flash: 256 KB RAM: 64 KB	70 × 83	Stop 2: 1,07 μA RX: 4,82 mA TX: 87 mA a 20 dBm	\$53,30 ST/distribuidores Ativo
ESP32-S3 + LR1121 [28]	Flash: 4 MB RAM: 512 KB PSRAM: 2 MB	41 × 21 × 3,2	Sono: não especificado RX: não especificado TX: 87,72 mA a 22 dBm	\$14,99–\$15,99 Waveshare Disponível
STM32L432KC + RA-02 [29, 35, 31]	Flash: 256 KB RAM: 64 KB	Não definida	Não medido	\$16,00 + RA-02 ST/distribuidores Ativo
LPC1769 + RA-02 [30, 36, 31]	Flash: 512 KB RAM: 64 KB	Não definida	Não medido	\$28,75 + RA-02 NXP Ativo e em stock

As dimensões são apresentadas em milímetros. Os consumos provêm da documentação dos fabricantes e referem-se às condições indicadas, não sendo diretamente comparáveis entre módulos e placas completas. Na NUCLEO-WL55JC, os valores referem-se apenas ao STM32WL55; a placa inclui ainda o programador ST-LINK, reguladores e LED, não existindo um valor agregado na documentação do fabricante. Os preços foram publicados pelo fabricante ou pela respetiva loja oficial e consultados em 10 de julho de 2026. São apresentados em dólares dos Estados Unidos, sem IVA, portes, antenas ou outros acessórios. A disponibilidade refere-se à mesma data.

Da análise efetuada, verifica-se que as soluções baseadas em **STM32WL** [20] continuam relevantes para versões otimizadas posteriores, devido à integração nativa de rádio LoRa e ao potencial de redução

da complexidade de hardware e do consumo energético. Contudo, o protótipo validado neste relatório utiliza controladores FreeRTOS STM32L432KC e LPC1769 com rádio LoRa externo RA-02/SX1278, opção adequada à disponibilidade de hardware e à demonstração de que a lógica do controlador não fica dependente de uma placa específica, desde que exista uma adaptação de baixo nível para a plataforma usada.

2.7 Posicionamento e Implicações para o Projeto

Face ao estado da arte analisado, o projeto posiciona-se como uma solução de controlo local para sistemas de rega, orientada à reutilização de infraestruturas existentes e à redução da complexidade de instalação.

A arquitetura proposta combina um concentrador baseado em ESP32-S3 [37], com interface Web local acessível por Wi-Fi, e controladores de válvula com comunicação LoRa em modo *peer-to-peer* (P2P) [11].

Esta opção elimina a necessidade de cablagem dedicada e evita a dependência de serviços externos, mantendo um compromisso adequado entre alcance, consumo energético e simplicidade de utilização. O sistema permite definir comandos de rega agendados, emitir comandos de controlo e operar autonomamente com base na configuração previamente definida.

Da análise efetuada, resultam as seguintes implicações para o desenvolvimento do protótipo:

- privilegiar operação autónoma e local, evitando dependência de serviços externos;
- equilibrar alcance e consumo energético em ambiente exterior;
- simplificar a instalação e a manutenção, reduzindo a necessidade de intervenção técnica especializada;
- garantir fiabilidade na entrega de comandos, recorrendo a mecanismos de confirmação e repetição sempre que necessário.

O protótipo implementado mantém o foco no controlo baseado em agenda, sendo os mecanismos de automação mais avançados, baseados em sensores ou dados externos, considerados como extensão posterior.

3 Requisitos e Arquitetura do Sistema

3.1 Requisitos Funcionais

Os requisitos funcionais definem as capacidades essenciais do sistema para garantir controlo local da rega, comunicação com controladores de válvula e visibilidade do estado operacional.

- **RF-01 — Gestão de controladores**: o sistema deve permitir adicionar, identificar, renomear, consultar e remover controladores de válvula. No protótipo, cada controlador corresponde a uma válvula;
- **RF-02 — Configuração de horários**: o utilizador deve poder criar, editar, ativar, suspender e eliminar horários de rega associados a cada controlador. Os horários são mantidos e avaliados no concentrador;
- **RF-03 — Base temporal**: o concentrador deve manter uma hora de referência para avaliação local dos horários, sem exigir que os controladores tenham hora de parede sincronizada;
- **RF-04 — Envio de comandos**: o concentrador deve entregar comandos de abertura e fecho aos controladores por comunicação LoRa ponto-a-ponto;
- **RF-05 — Confirmação de execução**: o controlador deve confirmar a receção e o resultado de cada comando por meio de `COMMAND_ACK`, tornando essa informação observável na API e na interface Web;
- **RF-06 — Retenção e nova oportunidade de entrega**: enquanto aguarda `COMMAND_ACK`, o concentrador deve manter um único comando em espera para o controlador. O controlador volta a contactar o concentrador após expiração de tempo, usando *jitter*/*backoff* aleatório limitado para reduzir colisões repetidas;
- **RF-07 — Operação autónoma local**: o concentrador deve executar os horários configurados sem depender de ligação contínua a dispositivos do utilizador ou a serviços externos.

3.2 Requisitos Não Funcionais

Os requisitos não funcionais estabelecem critérios de fiabilidade, continuidade de operação, usabilidade e eficiência adequados ao contexto residencial do protótipo.

- **RNF-01 — Fiabilidade da comunicação**: o protocolo deve permitir associar cada resposta de emparelhamento ao respetivo pedido e cada confirmação ao comando correspondente. Um comando não confirmado deve voltar a ser enviado num contacto posterior;
- **RNF-02 — Continuidade de operação**: após um reinício controlado, as configurações de rede, os controladores registados e os horários guardados devem coincidir com os valores anteriores ao reinício;
- **RNF-03 — Tempo de resposta**: com dois controladores registados, cada um de 100 pedidos consecutivos às rotas de consulta da API HTTP deve obter resposta em, no máximo, 1 s na rede local. A validação deve registar mediana, P95, P99 e máximo;

- **RNF-04 — Eficiência energética**: nas variantes de baixo consumo, para um intervalo de contacto igual ou superior a 5 s, o controlador deve entrar em WFI ou sono profundo entre duas janelas de comunicação consecutivas;
- **RNF-05 — Escalabilidade arquitetural**: o concentrador deve permitir o registo de, no mínimo, 32 controladores e manter, para cada um, o respetivo estado e a gestão de um comando pendente, sem alterações ao protocolo ou às estruturas de domínio;
- **RNF-06 — Operação local**: o sistema deve funcionar integralmente na rede local, sem serviços externos associados ou dependência de uma ligação à Internet. A interface Web deve permitir o emparelhamento e a consulta de controladores, a configuração de horários e o envio de comandos.

3.3 Cobertura no Protótipo

O protótipo integra concentrador ESP32-S3, firmware dos controladores STM32L432KC e LPC1769, módulos RA-02/SX1278 e comunicação LoRa real entre os dispositivos. O objetivo foi validar o ciclo funcional mínimo de ponta a ponta, mantendo o desenho preparado para extensão posterior.

Os aspetos cobertos pelo protótipo incluem:

- arranque do concentrador e dos serviços locais de armazenamento, rede, tempo, núcleo aplicacional, rádio e HTTP;
- interface Web e API para estado, emparelhamento, controladores, comandos, horários e eventos recentes;
- emparelhamento LoRa com correlação do pedido e registo concluído no primeiro `NODE_STATUS` válido;
- entrega de comandos por meio de `RESP_OPEN_VALVE` e `RESP_CLOSE_VALVE`;
- processamento de `COMMAND_ACK` pelo concentrador;
- criação e listagem de horários no concentrador;
- persistência local do estado aplicacional e das configurações do sistema.

Alguns requisitos permanecem delimitados ao nível de protótipo. Em particular, não foram realizados ensaios completos de alcance, autonomia, robustez em exterior ou operação prolongada com muitos controladores simultâneos. Também não é assumido endurecimento de segurança adequado a instalação de produção.

3.4 Validação dos Requisitos

A validação dos requisitos foi realizada por uma combinação de verificações de compilação, validação de firmware, testes funcionais com hardware e observação por meio da API e da interface Web. Foram considerados:

- compilação do firmware do concentrador em Espressif IoT Development Framework (ESP-IDF);
- validação de sintaxe e construção dos firmwares STM32L432KC/LPC1769 e dos módulos C portáveis do controlador;
- arranque do concentrador e dos controladores físicos;
- comunicação LoRa ponto-a-ponto com emparelhamento, estado, comando e confirmação;
- visibilidade dos controladores, comandos, horários e eventos por meio da API e da interface Web.

Esta validação confirma a coerência dos requisitos principais no protótipo implementado, sem substituir medições quantitativas prolongadas de alcance, autonomia, segurança ou escala.

3.5 Visão Geral da Arquitetura

A arquitetura do sistema segue um modelo distribuído do tipo *hub-and-spoke*, composto por um concentrador local e por controladores de válvula. O concentrador é implementado num ESP32-S3 e concentra a gestão do sistema, a persistência local, a interface de utilizador e a comunicação rádio. Cada controlador representa uma válvula e o protótipo validado usa firmware FreeRTOS executado em STM32L432KC e LPC1769, com ligação a um módulo RA-02/SX1278 para comunicação LoRa ponto-a-ponto.

A interação com o utilizador é realizada por meio de uma interface Web local, servida pelo próprio concentrador via Wi-Fi. Esta interface comunica com uma API HTTP local, permitindo observar controladores, abrir a janela de emparelhamento, enviar comandos de abertura ou fecho, configurar horários e consultar eventos recentes. Os horários residem no concentrador; os controladores não armazenam regras de agendamento, apenas executam comandos recebidos e reportam o seu estado.

O contrato de comunicação entre concentrador e controlador encontra-se concentrado no cabeçalho partilhado `protocol/include/protocol.h`. A arquitetura do protocolo assume um modelo simples: um controlador corresponde a uma válvula, o concentrador mantém no máximo um comando pendente por controlador e cada tipo de pacote LoRa possui um comprimento fixo.

A Figura 3.1 apresenta a visão global do sistema e das principais dependências entre os seus elementos.

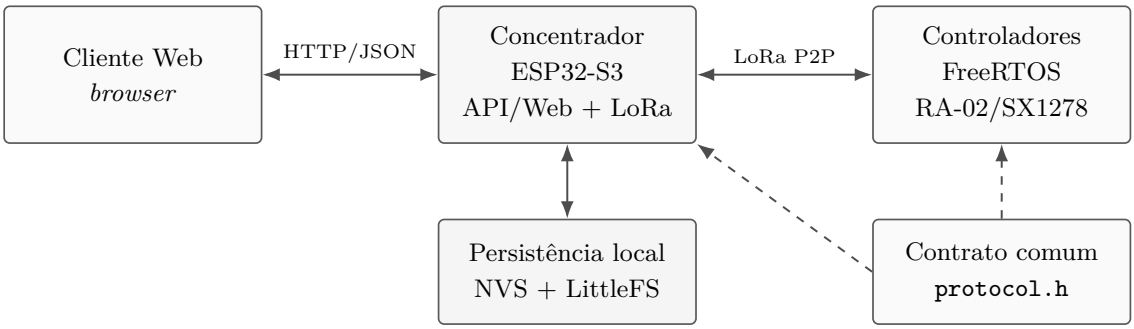


Figura 3.1: Visão global do sistema implementado.

A Tabela 3.1 sintetiza os blocos principais da solução.

Tabela 3.1: Blocos principais da solução

Bloco	**Responsabilidade principal**
Concentrador ESP32-S3	Disponibiliza a rede local, a interface Web, a API HTTP local, a persistência, a referência temporal e a ligação LoRa aos controladores.
Controladores STM32L432KC e LPC1769	Aplicam os comandos ao estado lógico representado pelos LEDs integrados das placas, mantêm estado local, suportam o emparelhamento e comunicam por LoRa com o concentrador.
Protocolo partilhado	Definição dos tipos de pacote, campos comuns, comandos e estados usados por ambos os firmwares.
Interface Web e API HTTP	Controlo e monitorização através da interface Web e das rotas da API HTTP.

3.6 Arquitetura Interna do Concentrador

O concentrador encontra-se organizado como um projeto ESP-IDF modular. A separação em componentes procura isolar responsabilidades e reduzir acoplamento entre domínio aplicacional, persistência, rede, interface Web, API HTTP e transporte rádio.

Os componentes principais são:

- **storage**: inicialização de NVS e LittleFS e acesso protegido aos mecanismos de persistência;
- **network_service**: configuração da rede Wi-Fi local, DNS local e anúncio do serviço por Multicast Domain Name System (mDNS);
- **time_service**: hora de parede, sincronização por Simple Network Time Protocol (SNTP) quando disponível e acerto manual;
- **app_core**: núcleo de domínio, responsável por controladores, emparelhamento, comandos, horários e eventos recentes;
- **http_api**: servidor HTTP, interface Web embebida e rotas da API HTTP;
- **lora_packet_codec**: codificação e validação dos pacotes binários LoRa;
- **radio_service**: tarefa de rádio que processa pacotes e interage com o **app_core**;
- **sx1278_driver**: adaptação ao módulo RA-02/SX1278, com gestão de SPI, GPIO e transições RX/TX.

No arranque, os serviços de suporte são inicializados antes do núcleo aplicacional e das interfaces HTTP e rádio, garantindo que a configuração e o estado persistente estão disponíveis antes do processamento de pedidos.

A Figura 3.2 resume a arquitetura interna do concentrador.

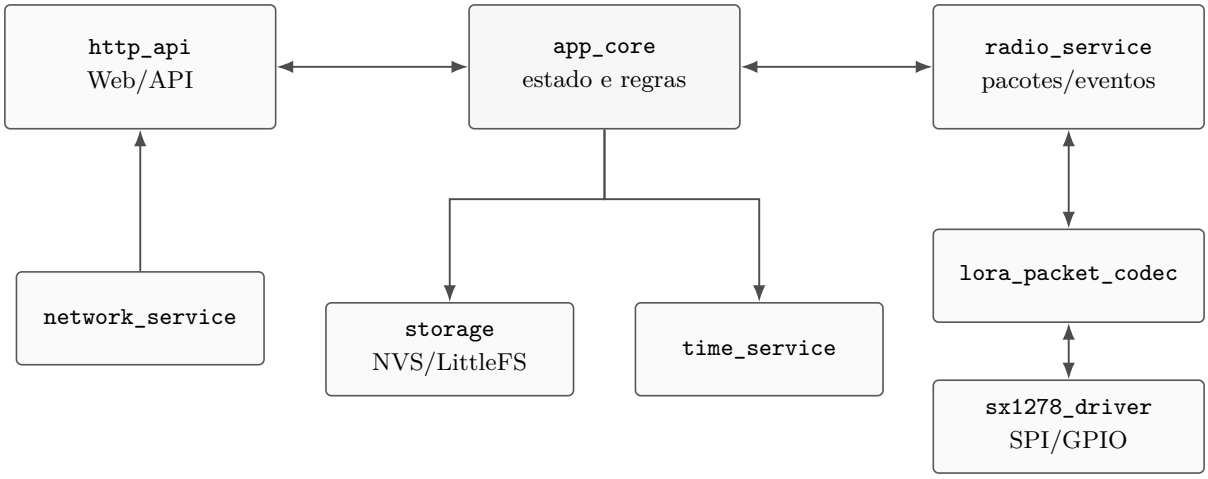


Figura 3.2: Arquitetura interna do concentrador e ligações principais entre componentes.

A Tabela 3.2 descreve as tarefas funcionais mais relevantes para a execução do concentrador.

Tabela 3.2: Tarefas e serviços principais do concentrador

Serviço/tarefa	**Componente**	**Função**
Interface Web e API HTTP	`http_api`	Serve a interface Web e processa operações locais sobre estado, emparelhamento, controladores, comandos, horários e eventos.
Núcleo aplicacional	`app_core`	Mantém a projeção de domínio, aplica regras de comando e horário, e persiste o *snapshot* do estado.
Gestão de horários	`app_core`	Verifica horários ativos e cria comandos de abertura ou fecho quando aplicável.
Serviço de rádio	`radio_service`	Mantém receção LoRa, trata pedidos dos controladores e transmite respostas através do controlador SX1278.
Serviços de suporte	`storage` rede e tempo	Fornecem persistência, conectividade local e base temporal para a restante aplicação.

3.7 Modelo de Estado, Comandos e Persistência

O `app_core` é o proprietário lógico do estado de domínio. Esta opção centraliza a aplicação de regras e evita que o servidor HTTP ou o serviço de rádio manipulem diretamente estruturas persistentes.

O modelo de estado segue as seguintes decisões:

- cada controlador corresponde a uma única válvula;
- o concentrador mantém o registo de controladores emparelhados e de inscrições em curso durante a janela de emparelhamento;
- existe no máximo um comando pendente por controlador;
- comandos manuais e comandos originados por horários são resolvidos pelo concentrador antes de serem entregues ao controlador;
- os horários residem apenas no concentrador e são avaliados localmente;
- eventos recentes são mantidos em memória, como histórico operacional imediato para a interface Web.

A persistência separa configuração estrutural, estado aplicacional e eventos transitórios. A configuração utiliza NVS, enquanto o estado durável do domínio é guardado em LittleFS. Os eventos recentes permanecem apenas em memória por terem finalidade diagnóstica.

3.8 Fluxos Operacionais Principais

3.8.1 Fluxo entre Interface Web e Concentrador

O fluxo local de utilização é realizado sobre HTTP e JSON. A interface Web servida pelo concentrador invoca rotas da API HTTP, que por sua vez encaminham operações para o `app_core`. Este fluxo cobre consulta de estado, gestão da janela de emparelhamento, listagem e renomeação de controladores, criação de comandos, configuração de horários e consulta de eventos recentes.

3.8.2 Fluxo de Emparelhamento e Estado do Controlador

O emparelhamento é iniciado pelo controlador por meio de `PAIR_REQUEST`, contendo um valor temporário de correlação, designado por *nonce*. O `radio_service` entrega o pedido ao `app_core`, que aceita ou rejeita a inscrição consoante o estado da janela de emparelhamento. A resposta `PAIR_ACCEPT` ou `PAIR_REJECT` ecoa esse valor, permitindo ao controlador validar que a resposta corresponde à tentativa atual.

Após aceitação, o registo fica concluído quando o concentrador recebe o primeiro `NODE_STATUS` válido do identificador atribuído. A partir daí, cada estado recebido atualiza a projeção do controlador e permite ao concentrador devolver uma resposta fixa: `RESP_EMPTY`, `RESP_CONFIG`, `RESP_OPEN_VALVE` ou `RESP_CLOSE_VALVE`.

3.8.3 Fluxo de Comando

Os comandos de abertura ou fecho podem ser criados pela interface Web ou pela avaliação de horários. O `app_core` mantém apenas um comando pendente por controlador e o `radio_service` inclui esse comando numa resposta ao próximo contacto do controlador. Após executar a ação, o controlador transmite `COMMAND_ACK`; o concentrador usa essa confirmação para resolver o comando e atualizar os eventos recentes.

Os comandos de abertura deixam de ser reenviados quando termina o respetivo prazo de validade ou após três envios sem confirmação. O concentrador regista-os então como expirados e remove-os do estado

pendente. Os comandos de fecho continuam a ser enviados em contactos posteriores até à receção do ACK, por conduzirem o sistema para o estado de segurança.

A Figura 3.3 sintetiza o ciclo de interação em *runtime*.

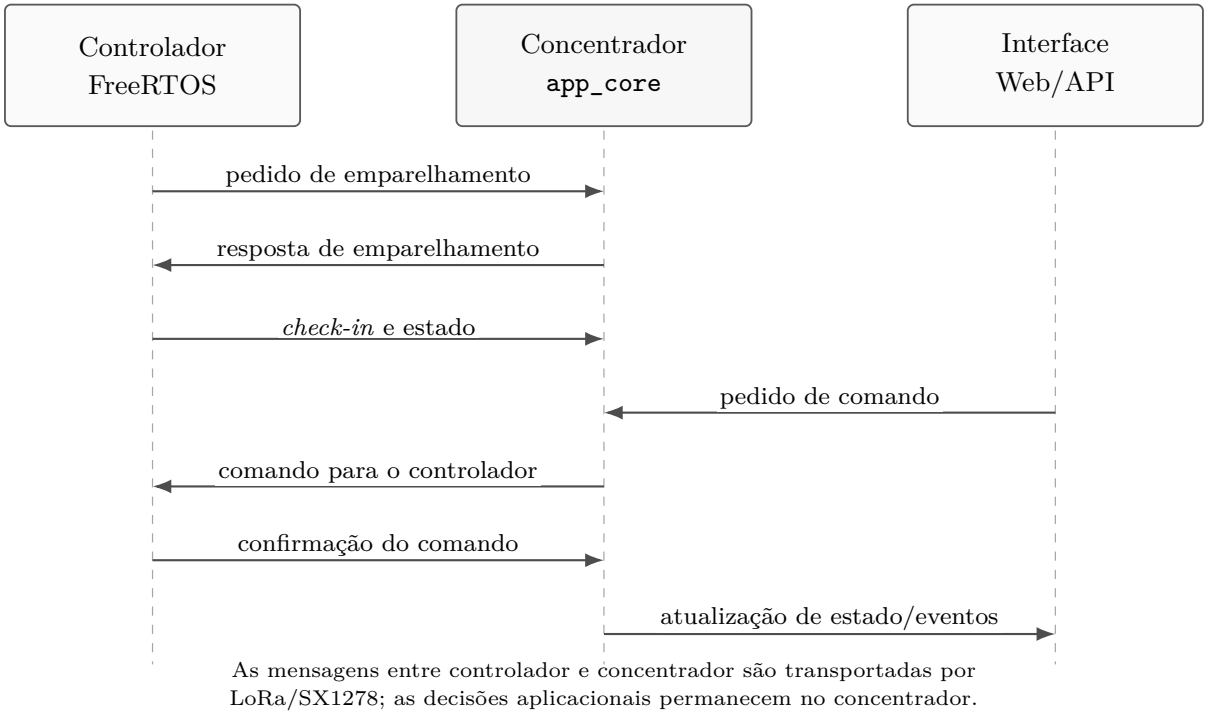


Figura 3.3: Ciclo de interação em *runtime* entre controlador, concentrador e interface Web.

3.9 Arquitetura do Protocolo

O protocolo de comunicação encontra-se definido em `protocol/include/protocol.h` e é usado tanto pelo concentrador como pelo firmware comum do controlador. A codificação dos pacotes LoRa é realizada em camada própria, com campos explícitos de tamanho fixo e inteiros em *little-endian*. Todos os pacotes começam por um byte de tipo e o descodificador só aceita o comprimento exato esperado para esse tipo de pacote.

A decisão principal foi manter o protocolo pequeno e fácil de observar. Por isso, os identificadores de controlador e de comando têm um byte na comunicação rádio, não existe byte de versão, não existe campo de comprimento, não existe checksum aplicacional e não são transportados campos de hora do servidor, expiração de comando ou versão de configuração. A integridade básica fica a cargo do CRC do próprio modo LoRa.

A Tabela 3.3 identifica os tipos de pacote, o sentido da transmissão e o comprimento total. O campo de tipo ocupa sempre o byte 0 e tem um byte de largura; o valor `0x03` permanece reservado.

Tabela 3.3: Pacotes, códigos e comprimentos do protocolo binário partilhado

Pacote	**Sentido**	**Tipo (byte 0)**	**Bytes**
`PAIR_REQUEST`	Controlador → concentrador	`0x00`	5 bytes
`PAIR_ACCEPT`	Concentrador → controlador	`0x01`	10 bytes
`PAIR_REJECT`	Concentrador → controlador	`0x02`	8 bytes
`NODE_STATUS`	Controlador → concentrador	`0x04`	6 bytes
`COMMAND_ACK`	Controlador → concentrador	`0x05`	4 bytes
`RESP_EMPTY`	Concentrador → controlador	`0x06`	4 bytes
`RESP_CONFIG`	Concentrador → controlador	`0x07`	8 bytes
`RESP_OPEN_VALVE`	Concentrador → controlador	`0x08`	7 bytes
`RESP_CLOSE_VALVE`	Concentrador → controlador	`0x09`	5 bytes

A Tabela 3.4 mostra a posição de cada campo nos pacotes, enquanto a Tabela 3.5 reúne a largura, a unidade, a gama válida e o significado de cada campo sem repetir definições. As posições são contadas a partir de zero e os inteiros com mais de um byte usam a ordem *little-endian*. Cada tipo de pacote tem comprimento fixo, embora os comprimentos sejam diferentes entre tipos.

Tabela 3.4: Posição dos campos em cada tipo de pacote

Pacote	**Posição dos campos, em bytes**
`PAIR_REQUEST`	0: `type`; 1–4: `request_nonce`.
`PAIR_ACCEPT`	0: `type`; 1–4: `request_nonce`; 5: `assigned_node_id`; 6–7: `normal_checkin_s`; 8–9: `active_checkin_s`.
`PAIR_REJECT`	0: `type`; 1–4: `request_nonce`; 5–6: `retry_after_s`; 7: `reason`.
`NODE_STATUS`	0: `type`; 1: `node_id`; 2: `valve_state`; 3: `flags`; 4: `last_command_id`; 5: `last_command_result`.
`COMMAND_ACK`	0: `type`; 1: `node_id`; 2: `command_id`; 3: `result`.
`RESP_EMPTY`	0: `type`; 1: `node_id`; 2–3: `next_checkin_s`.
`RESP_CONFIG`	0: `type`; 1: `node_id`; 2–3: `normal_checkin_s`; 4–5: `active_checkin_s`; 6–7: `next_checkin_s`.
`RESP_OPEN_VALVE`	0: `type`; 1: `node_id`; 2: `command_id`; 3–4: `duration_s`; 5–6: `next_checkin_s`.
`RESP_CLOSE_VALVE`	0: `type`; 1: `node_id`; 2: `command_id`; 3–4: `next_checkin_s`.

Tabela 3.5: Largura, gama e significado dos campos do protocolo

Campo	Bytes	Unid.	Gama válida	Significado
`type`	1	—	`0x00`–`0x09`; `0x03` reservado	Identifica o tipo de pacote, de acordo com a Tabela 3.3.
`request_nonce`	4	—	1–4 294 967 295	Identifica a tentativa de emparelhamento e correlaciona o pedido com a resposta.
`node_id` / `assigned_node_id`	1	—	1–255	Identifica o controlador ou o valor que lhe é atribuído.
`normal_checkin_s`	2	s	3–65 535	Intervalo de contacto quando a saída está fechada.
`active_checkin_s`	2	s	3–65 535	Intervalo de contacto quando a saída está aberta.
`retry_after_s`	2	s	0–3 600	Tempo mínimo antes de uma nova tentativa de emparelhamento.
`reason`	1	—	0–3	Motivo da rejeição: 0 fechado; 1 ocupado; 2 registo cheio; 3 protocolo não suportado.
`valve_state`	1	—	0–5	Estado lógico: 0 desconhecido; 1 aberto; 2 fechado; 3 a abrir; 4 a fechar; 5 falha.
`flags`	1	—	`0x00`, `0x01`	O bit 0 indica bateria fraca; os restantes bits devem ser zero.
`last_command_id`	1	—	0–255	Último comando aplicado; zero indica ausência de comando anterior.
`last_command_result`	1	—	0–4	Resultado do último comando: 0 nenhum; 1 sucesso; 2 falha; 3 rejeitado; 4 expirado.
`command_id`	1	—	1–255	Identifica um comando de abertura ou fecho.
`result`	1	—	1–4	Resultado confirmado: 1 sucesso; 2 falha; 3 rejeitado; 4 expirado.
`next_checkin_s`	2	s	3–65 535	Intervalo até ao contacto seguinte.
`duration_s`	2	s	1–21 600	Duração da abertura, limitada a seis horas.

A Tabela 3.6 apresenta exemplos serializados com o identificador de controlador 1, o *nonce* `0x12345678`, intervalos de 60 s e 5 s, comandos 42 e 43 e uma duração de abertura de 300 s. O estado exemplificado corresponde à saída fechada, sem *flags* nem comando anterior, e o ACK indica sucesso.

Tabela 3.6: Exemplos hexadecimais de pacotes do protocolo

Pacote	Bytes serializados, em hexadecimal
`PAIR_REQUEST`	`00 78 56 34 12`
`PAIR_ACCEPT`	`01 78 56 34 12 01 3C 00 05 00`
`NODE_STATUS`	`04 01 02 00 00 00`
`RESP_OPEN_VALVE`	`08 01 2A 2C 01 05 00`
`COMMAND_ACK`	`05 01 2A 01`
`RESP_CLOSE_VALVE`	`09 01 2B 3C 00`

Nos campos com mais de um byte, a ordem *little-endian* coloca primeiro o byte menos significativo. Por exemplo, o *nonce* `0x12345678` é serializado como `78 56 34 12`.

O valor 0 está reservado para representar um identificador inválido ou ainda não atribuído. Na implementação atual, os identificadores de controlador são atribuídos entre 1 e 32; quando esse conjunto está ocupado, o emparelhamento é rejeitado, pelo que não existe transição de 255 para 0. Os identificadores de comando utilizam os valores de 1 a 255. Após 255, a atribuição recomeça em 1. Se o valor considerado já identificar um comando pendente, o concentrador avança até encontrar um identificador livre.

A arquitetura não inclui uma mensagem `PAIR_CONFIRM` nem um comando `RETRY_LATER`. A conclusão do emparelhamento decorre do primeiro `NODE_STATUS` válido, e a contenção é mitigada por temporizações, expirações e *jitter*/*backoff* aleatório limitado nos controladores. As respostas de comando foram separadas em tipos próprios para que uma resposta sem comando, `RESP_EMPTY`, não transporte campos desnecessários.

3.10 Firmware do Controlador de Válvula

A arquitetura do controlador divide-se em duas camadas. A primeira é portável e reside em `node-firmware/common`, com módulos C para protocolo, estado do controlador, emparelhamento, temporização, aplicação de comandos e interface de portabilidade. A segunda camada corresponde ao código específico da placa, responsável por SPI, GPIO, temporização, rádio RA-02/SX1278, persistência local e indicação da saída simulada.

Foram implementadas versões FreeRTOS para duas placas: STM32L432KC e LPC1769. Ambas usam a mesma lógica comum de controlador e apenas substituem a camada de portabilidade. Esta separação permite reutilizar o comportamento de emparelhamento, verificação, receção de comandos e confirmação, mantendo as diferenças de hardware isoladas.

Nas duas placas, o controlador persiste o identificador atribuído, os intervalos de verificação, o último comando aplicado, o respetivo resultado e o estado lógico da saída simulada. O estado do comando é guardado antes do envio de `COMMAND_ACK`; se o mesmo comando voltar a ser entregue após uma reinicialização, o resultado anterior é retransmitido sem repetir a atuação.

Após uma reinicialização completa, o estado é restaurado e é executada uma única operação de fecho seguro, ficando também persistido o estado lógico fechado. Os despertares normais dos modos de baixo consumo não desencadeiam esse fecho. Isto privilegia a segurança, embora uma reinicialização durante o período de abertura possa interromper a rega antes do tempo previsto. Sem um sensor de posição, o estado lógico fechado regista a operação de fecho, mas não confirma a posição física.

3.11 Decisões Arquiteturais Relevantes

Ao longo do desenvolvimento foram adotadas decisões orientadas à simplicidade e à verificabilidade do protótipo:

- **núcleo de domínio centralizado**: o `app_core` concentra o estado e as regras de decisão, evitando duplicação entre HTTP e rádio;
- **persistência local mínima**: configurações pequenas ficam em NVS e o estado aplicacional durável fica num *snapshot* binário em LittleFS;
- **protocolo curto**: cada contacto do controlador permite receber estado e devolver uma resposta fixa, sem campos de comando quando não há comando;
- **operação local sem serviços externos**: o sistema é operável na rede local ou no ponto de acesso do concentrador;
- **horários no concentrador**: o controlador não armazena agendas, reduzindo a complexidade e o estado persistente necessário em cada válvula;

- **contenção por temporização**: novas tentativas e verificações periódicas usam atrasos aleatórios limitados, adequados à escala do protótipo.

3.12 Modos de Conectividade

O concentrador foi desenhado para funcionar de forma local e manter o sistema utilizável mesmo sem infraestrutura externa. Em modo AP, cria a sua própria rede Wi-Fi para instalação ou recuperação. Em modo STA, entra numa rede existente e pode ser acedido por endereço IP ou por `irrigation.local`, quando o cliente suporta resolução local por DNS ou mDNS. A Figura 3.4 sintetiza estes dois modos.

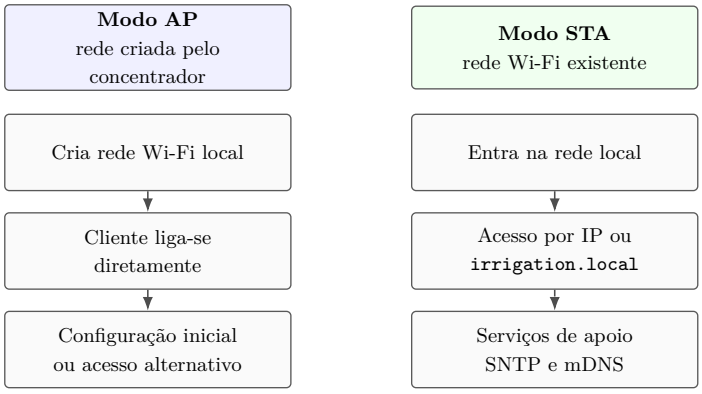


Figura 3.4: Modos de conectividade suportados pelo concentrador.

3.13 Cenários de Utilização

A arquitetura também pode ser lida a partir dos cenários de utilização. Estes cenários agrupam as ações principais do utilizador: acesso ao sistema, monitorização, envio manual de comandos de abertura ou fecho e configuração. A Figura 3.5 apresenta esse mapa de forma resumida.

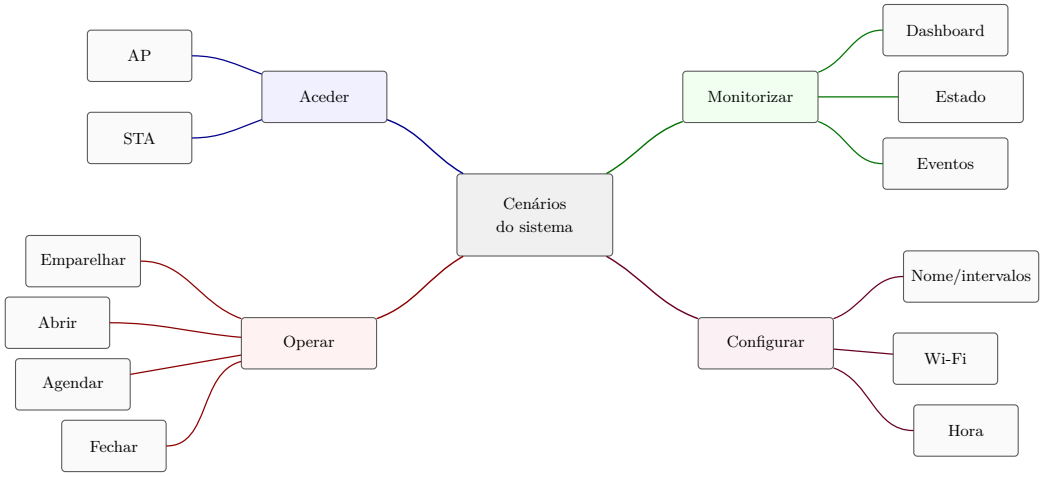


Figura 3.5: Mapa resumido dos cenários de utilização suportados pela arquitetura atual.

3.14 Limitações Arquiteturais da Iteração Atual

A arquitetura atual privilegia simplicidade, operação local e validação do ciclo funcional principal. As fronteiras arquiteturais mais relevantes são a operação LoRa ponto-a-ponto sem coordenação temporal rígida, a ausência de serviços externos de sincronização aplicacional, a segurança local ainda simples

e a ausência de realimentação física sobre a posição da válvula. A análise detalhada de limitações experimentais, autonomia, alcance, escala e segurança é reservada ao capítulo final do relatório.

3.15 Síntese

A arquitetura atual descreve um protótipo integrado: concentrador ESP32-S3, controladores STM32L432KC e LPC1769, comunicação RA-02/SX1278, protocolo partilhado, interface Web, API HTTP e persistência embarcada. A separação entre `app_core`, `http_api`, `radio_service`, codec, controlador SX1278 e serviços de suporte estabelece uma base coerente para a implementação e validação descritas nos capítulos seguintes.

4 Implementação

4.1 Implementação do Concentrador

4.1.1 Organização em Componentes

O firmware do concentrador encontra-se organizado como um projeto ESP-IDF modular, com fronteiras explícitas entre persistência, rede, tempo, domínio aplicacional, HTTP e rádio. Os componentes principais são:

- `storage`: inicializa e valida os mecanismos locais de persistência, expondo primitivas síncronas e protegidas por mutex sobre NVS e LittleFS;
- `network_service`: gere Wi-Fi em modo AP+STA, configuração de rede, DNS local para o ponto de acesso e anúncio `irrigation.local` por mDNS;
- `time_service`: fornece hora de parede para horários, com sincronização por SNTP quando disponível e acerto manual através da interface Web;
- `app_core`: concentra o estado de domínio, incluindo registo de controladores, janela de emparelhamento, comandos, horários e eventos recentes;
- `http_api`: serve a interface Web embebida e as rotas da API HTTP em `/api`;
- `lora_packet_codec`: codifica, descodifica e valida os pacotes binários usados na comunicação LoRa;
- `radio_service`: executa a tarefa de rádio, traduzindo pacotes recebidos em eventos semânticos para o `app_core`;
- `sx1278_driver`: encapsula SPI, GPIO e a biblioteca SX127x usada para operar o módulo RA-02/SX1278.

O arranque é coordenado em `app_main.c`. A sequência inicializa armazenamento, rede, tempo, núcleo aplicacional, tarefa periódica de avaliação de horários, serviço de rádio e, por fim, servidor HTTP. Os serviços seguem um ciclo de vida *start-once/run-forever*: uma vez inicializados, permanecem ativos durante a execução do firmware, e chamadas de arranque repetidas são tratadas como estado inválido.

4.1.2 Persistência Local

A persistência local é deliberadamente separada por tipo de dado. A NVS usa o *namespace* `system` para configurações pequenas e duráveis, como parâmetros de rede, nome local e opções associadas ao tempo. O LittleFS é montado em `/littlefs` e armazena o estado durável do núcleo aplicacional.

O `app_core` persiste esse estado num *snapshot* binário compacto na área `/littlefs/state/`, com o ficheiro `app_core_snapshot_v1.bin`. O *snapshot* contém controladores conhecidos ou em registo pendente, nomes de apresentação, estado de comandos pendentes, regras de horário e contadores de identificação. Eventos recentes são mantidos apenas em memória, por servirem de histórico operacional imediato para a interface Web e não de registo durável.

4.2 Núcleo da Aplicação

O `app_core` é a fronteira de domínio do concentrador. Os pedidos HTTP, os eventos de rádio e a tarefa periódica de horários convergem neste módulo, evitando que a interface Web ou o serviço de rádio manipulem diretamente estruturas persistentes.

As responsabilidades implementadas incluem:

- **registo de controladores**: manutenção de controladores emparelhados, estado lógico da saída simulada, último contacto, indicação de bateria fraca e nome visível na interface;
- **emparelhamento**: abertura temporária de uma janela de emparelhamento, criação de uma inscrição pendente e associação do *nonce* recebido no `PAIR_REQUEST`;
- **horários**: regras locais por controlador, ativáveis ou suspensas, que são avaliadas periodicamente e convertidas em comandos de abertura ou fecho;
- **comandos**: existência de um único comando pendente por controlador, resolvido quando chega o respetivo `COMMAND_ACK` ou substituído apenas em casos controlados, como fecho manual ou de segurança;
- **eventos recentes**: anel em memória com alterações relevantes, exposto à interface Web para observação rápida do comportamento do sistema.

No emparelhamento, o concentrador responde ao pedido inicial com aceitação ou rejeição, repetindo o mesmo *nonce* para que o controlador ignore respostas de outras tentativas. A associação fica concluída quando o primeiro `NODE_STATUS` válido é recebido a partir do identificador atribuído. O firmware ativo não emite `PAIR_CONFIRM` nem o comando `RETRY_LATER`; a gestão de contenção é realizada por temporização e atrasos aleatórios limitados nos controladores.

4.3 Implementação LoRa e Rádio

A camada LoRa divide-se em três níveis. O cabeçalho partilhado `protocol/include/protocol.h` define os valores numéricos comuns a ambos os firmwares. O componente `lora_packet_codec` implementa a serialização determinística, sem alocação dinâmica, usando campos de tamanho fixo em *little-endian*. O `radio_service` integra esse codec com o controlador SX1278 e com o `app_core`.

A descrição canónica da serialização encontra-se nas Tabelas 3.3, 3.4 e 3.5. Cada tipo de pacote LoRa possui um comprimento fixo, mas os comprimentos diferem entre tipos. Como os valores incluem o byte de tipo, o comprimento observado no rádio identifica o formato esperado.

O driver `sx1278_driver` encapsula a configuração do módulo RA-02/SX1278, incluindo SPI, GPIO de interrupção e estados RX/TX. A configuração do protótipo usa 433,920 MHz, largura de banda de 125 kHz, *spreading factor* 9, taxa de codificação 4/5, preâmbulo de 8 símbolos, *sync word* `0x12`, cabeçalho explícito e CRC ativo.

Em execução, o `radio_service` mantém o rádio em receção contínua e processa três fluxos principais:

- `PAIR_REQUEST`: o pedido é entregue ao `app_core`, que decide aceitar ou rejeitar a inscrição; a resposta ecoa o *nonce* recebido;
- `NODE_STATUS`: o estado do controlador é convertido num evento de domínio e, enquanto o controlador permanece acordado, o concentrador devolve uma resposta fixa do protocolo, como `RESP_EMPTY`, `RESP_CONFIG` ou uma resposta de comando;
- `COMMAND_ACK`: o resultado do comando é registado no `app_core` e o rádio regressa ao modo de receção.

O protocolo não transporta hora do servidor, versão de configuração ou expiração de comando. A ausência desses campos reduz o tempo no ar e simplifica a implementação dos controladores. Como os

controladores contactam periodicamente o concentrador e recebem comandos apenas nesse momento, a entrega continua a ser controlada pelo estado pendente mantido no concentrador.

4.4 *Pinout* e Ligações de Hardware

As ligações do rádio foram mantidas explícitas no firmware para facilitar a reprodução e o diagnóstico. Todos os módulos RA-02/SX1278 devem ser alimentados a 3,3 V, com GND comum e antena ligada antes de transmitir. As tabelas seguintes mantêm o mapeamento exato dos pinos de cada montagem.

Tabela 4.1: *Pinout* RA-02/SX1278 no concentrador ESP32-S3

RA-02/SX1278	**ESP32-S3**	**Função**
VCC	3V3	Alimentação do módulo
GND	GND	Referência comum
SCK	GPIO10	SPI clock
MISO	GPIO11	SPI MISO
MOSI	GPIO12	SPI MOSI
NSS/CS	GPIO13	Seleção SPI
RESET	GPIO16	Reset do rádio
DIO0	GPIO17	IRQ RX/TX done
DIO1	GPIO18	IRQ auxiliar
DIO2	GPIO8	IRQ auxiliar

Tabela 4.2: *Pinout* RA-02/SX1278 no controlador STM32L432KC

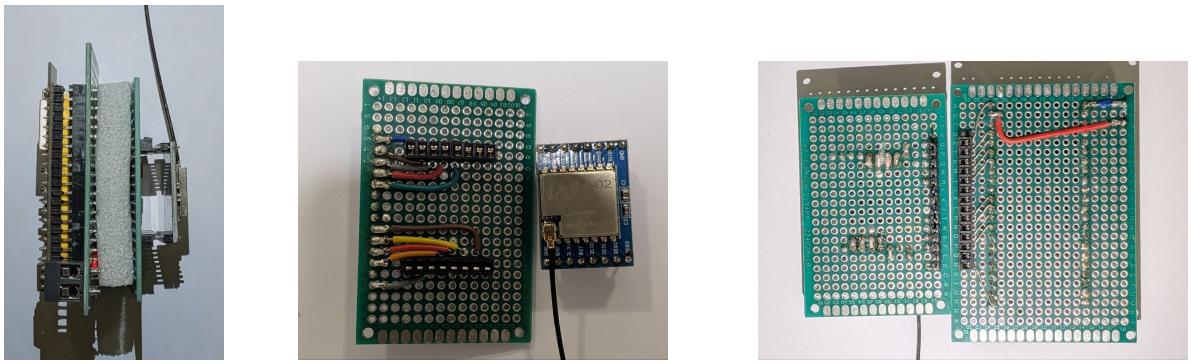
RA-02/SX1278	**NUCLEO-L432KC**	**Função**
VCC	3V3	Alimentação do módulo
GND	GND	Referência comum
SCK	D13 / PB3	SPI1 clock
MISO	D12 / PB4	SPI1 MISO
MOSI	D11 / PB5	SPI1 MOSI
NSS/CS	D10 / PA11	Seleção manual por GPIO
RESET	D2 / PA12	Reset do rádio
DIO0	D3 / PB0	IRQ RX/TX done
Indicação da saída simulada	LED LD3 integrado da placa	LED ligado representa estado lógico aberto

Tabela 4.3: *Pinout* RA-02/SX1278 no controlador LPC1769

RA-02/SX1278	**LPCXpresso LPC1769**	**Função**
VCC	3V3	Alimentação do módulo
GND	GND	Referência comum
SCK	J6-7 / P0.7 / SCK1	SSP1 clock
MISO	J6-6 / P0.8 / MISO1	SSP1 MISO
MOSI	J6-5 / P0.9 / MOSI1	SSP1 MOSI
NSS/CS	J6-8 / P0.6	Seleção manual por GPIO
RESET	J6-10 / P0.21	Reset do rádio
DIO0	J6-9 / P2.13	IRQ RX/TX done
Indicação da saída simulada	LED integrado da placa	LED ligado representa estado lógico aberto

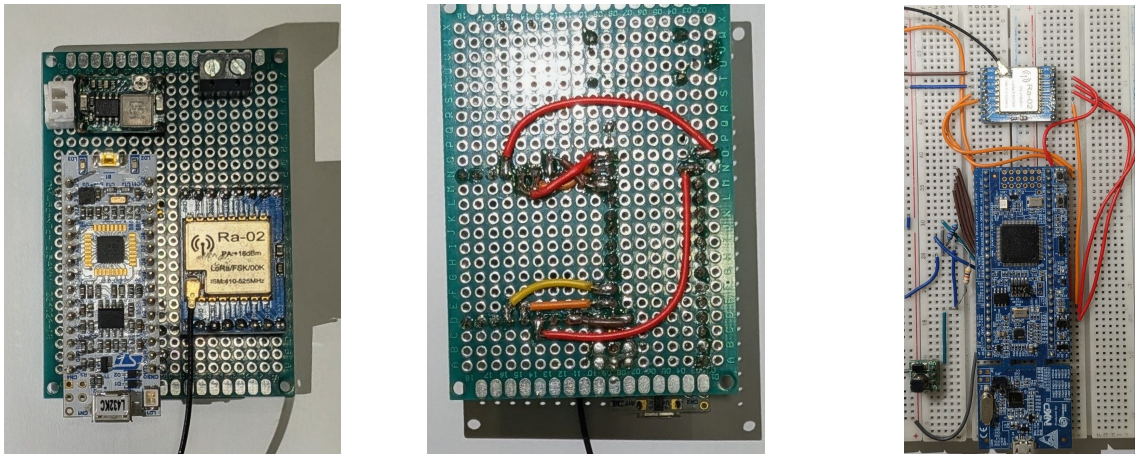
4.4.1 Montagens Construídas

As Figuras 4.1 e 4.2 apresentam as montagens reais usadas na validação. As fotografias complementam os mapeamentos elétricos anteriores e mostram a integração das placas com os módulos RA-02/SX1278.



(a) (b) (c)

Figura 4.1: Montagem do concentrador ESP32-S3: (a) conjunto empilhado; (b) placa adaptadora e módulo RA-02; (c) ligações soldadas nas placas perfuradas.



(a) (b) (c)

Figura 4.2: Montagens dos controladores: (a) frente da montagem STM32L432KC; (b) verso da montagem STM32L432KC; (c) montagem LPC1769 em placa de ensaio.

4.5 Firmware do Controlador de Válvula

O firmware do controlador de válvula foi organizado em código C portável e em camadas de portabilidade específicas para STM32L432KC e LPC1769. A parte reutilizável encontra-se em `node-firmware/common` e inclui:

- `node_protocol.c` e `node_protocol.h`, responsáveis pela codificação e descodificação dos pacotes definidos em `protocol/include/protocol.h`;
- `node.c` e `node.h`, que implementam auxiliares de estado, emparelhamento, cálculo de intervalos, aplicação de comandos e fecho automático da saída simulada;
- `node_port.h`, que define *callbacks* de plataforma para tempo, atrasos, rádio, válvula, bateria, aleatoriedade e persistência.

Em ambas as placas, o ciclo principal corre sobre o escalonador FreeRTOS. A tarefa `node_app_task` concentra o emparelhamento, o envio periódico de estado, a receção de respostas do concentrador e a

aplicação de comandos. Os tempos de espera são encaminhados pela camada de plataforma e usam `vTaskDelay()` nas variantes base, permitindo que a lógica comum seja reutilizada nas duas placas.

A separação por *callbacks* em `node_port.h` mantém a máquina de estados do controlador independente dos detalhes de hardware. Assim, o mesmo código de protocolo e de *runtime* é usado nas duas placas, ficando nas camadas de portabilidade específicas o acesso a SPI/SSP, GPIO, armazenamento local, aleatoriedade e modos de baixo consumo. O *idle hook* do FreeRTOS executa `wfi` quando não há trabalho ativo; nas variantes de sono profundo, o atraso entre contactos pode ser substituído por uma suspensão com despertar por RTC.

A placa STM32L432KC usa SPI1 para o RA-02 e a placa LPC1769 usa SSP1. Nas duas placas, a saída de atuação simulada é representada pelo LED integrado da placa; as camadas de placa normalizam o comportamento para que o LED visivelmente ligado represente o estado lógico aberto.

Nas duas placas, se não existir identidade persistida válida, o controlador envia `PAIR_REQUEST` com *nonce* não nulo até ser aceite pela janela de emparelhamento do concentrador. Após receber `PAIR_ACCEPT`, o controlador envia `NODE_STATUS` para concluir o emparelhamento no concentrador e guarda o identificador e os intervalos de verificação. Durante a operação, trata respostas vazias, atualizações de configuração e comandos de abertura ou fecho, respondendo aos comandos com `COMMAND_ACK`.

4.5.1 Persistência e Comportamento após Reinicialização

O estado persistido pelos controladores inclui a identidade, os intervalos de verificação normal e ativo, o estado lógico da saída simulada, o identificador do último comando aplicado e o respetivo resultado. O registo contém versão e *checksum*; registos inválidos ou de versões anteriores são rejeitados. No STM32L432KC é usada a última página da memória Flash, enquanto no LPC1769 é usado o setor 29 através da interface IAP do fabricante.

Após aplicar um comando, o controlador guarda o novo estado antes de enviar `COMMAND_ACK`. Deste modo, uma repetição do comando após perda da confirmação ou reinicialização devolve o resultado persistido sem voltar a atuar a saída. Em cada arranque completo é executado um único fecho seguro e o estado fechado é novamente guardado. Este tratamento não é executado nos despertares normais das variantes de baixo consumo.

4.6 Atuador de Válvula e Estimativa Energética

4.6.1 Dimensionamento do solenoide e do driver

A iteração implementada valida a cadeia de comando, estado e confirmação, mas o acionamento eletromecânico real da válvula foi deliberadamente abstraído. No STM32L432KC e no LPC1769, a indicação da saída de atuação é feita pelo LED integrado da placa; em ambos os casos o firmware chama a interface `valve_port` como se existisse um atuador. Assim, os valores desta secção são estimativas de dimensionamento para uma revisão de hardware, não resultados medidos de um driver final.

A documentação Rain Bird TBOS-BT identifica controladores de 9 V a pilha compatíveis com o solenoide de trinco TBOS/K80920 [38, 39]. Ao contrário de um solenoide convencional, este tipo de atuador só consome energia durante um pulso de abertura ou fecho e mantém a posição sem corrente contínua. Para a válvula considerada no projeto, o modelo elétrico usado foi uma bobina de cerca de 50 Ω acionada a 9 V. O requisito imediato para o driver é, portanto, de corrente e potência de pico:

$$I = \frac{V}{R} = \frac{9}{50} \approx 0{,}18 \text{ A} \tag{4.1}$$

$$P = \frac{V^2}{R} = \frac{9^2}{50} \approx 1{,}62 \text{ W} \tag{4.2}$$

Para discutir autonomia, o mesmo pulso tem de ser convertido em energia, porque a bateria não fornece 1,62 W continuamente; fornece essa potência apenas durante algumas dezenas ou centenas de

milissegundos. Para um pulso conservador de 100 ms:

$$E_{pulso} = Pt \approx 1{,}62 \times 0{,}1 = 0{,}162 \text{ J} \approx 0{,}045 \text{ mWh} \tag{4.3}$$

Com uma eficiência de 80% no caminho de alimentação/driver, isto corresponde a cerca de 0,056 mWh retirados da bateria por pulso, ou 0,112 mWh para uma sequência abrir+fechar. O valor exato do pulso deve ser confirmado no datasheet e por testes com a válvula final, porque a corrente na bobina é limitada pela indutância e pode depender de temperatura, pressão hidráulica, orientação da válvula e queda de tensão da bateria.

A Tabela 4.4 resume as implicações de diferentes classes de atuador. A diferença principal é que solenoides não biestáveis precisam de potência de retenção enquanto a válvula está aberta. Por exemplo, as válvulas Rain Bird CP/CPF de 24 VAC indicam 0,41 A/9,9 VA de arranque, 0,14 A/3,43 VA de retenção e resistência de bobina de 30–39 Ω [40]. Uma hora de rega com retenção de vários VA já consome energia da ordem de Wh, várias ordens de grandeza acima dos pulsos de um solenoide de trinco.

Tabela 4.4: Comparação de classes de solenoide para um controlador de rega alimentado por bateria

Atuador	**Exigência elétrica**	**Adequação ao protótipo**
9 V DC biestável/*latching*	Pulso curto com inversão de polaridade; no modelo usado, 0,18 A e 1,62 W durante dezenas a centenas de ms.	Melhor opção para controladores a bateria: energia baixa por atuação, mas requer ponte H, geração de 9 V ou reserva capacitiva e proteção indutiva.
12 V DC biestável/*latching*	Princípio semelhante, mas com maior tensão de acionamento e energia dependente da resistência da bobina.	Viável, mas aumenta a tensão exigida ao circuito de atuação. Deve ser redimensionado para a bobina escolhida.
24 VAC convencional	Necessita corrente de arranque e corrente de retenção enquanto a válvula permanece aberta.	Pouco adequado para bateria se a rega durar minutos: a potência de retenção domina a energia diária.
Solenoide DC não biestável	Driver simples, mas exige corrente contínua para manter a válvula numa posição.	Apenas aceitável se os tempos de abertura forem muito curtos ou se existir alimentação externa.

4.6.2 Estimativa de consumo em função do intervalo de despertar

A estimativa energética considera a eletrónica a 3,3 V, o SX1278 a cerca de 100 mA em TX a +17 dBm, 12 mA em RX, uma janela de receção bem-sucedida de 250 ms e tempo no ar LoRa de 123,9 ms para os pacotes curtos do protocolo [31]. Para comparar perfis, os resultados são expressos como energia diária retirada de uma bateria Li-ion de 3,7 V, com 85% de eficiência no regulador da eletrónica e 80% no caminho do solenoide.

Foram consideradas as variantes resumidas na Tabela 4.5. A distinção principal é entre as variantes com WFI, em que o processador fica em espera por interrupção durante períodos sem trabalho ativo, e as variantes com sono profundo, que reduzem o consumo entre contactos periódicos.

Tabela 4.5: Variantes de firmware consideradas na análise energética

Variante	**Estado**	**Notas**
STM32L432KC com WFI	Implementada e validada	Versão base usada na validação funcional inicial; usa WFI em *idle*, mantendo o SX1278 em *standby* entre contactos.
LPC1769 com WFI	Implementada e validada	Versão base validada para a placa LPC1769; maior consumo de base no modelo energético.
STM32L432KC com sono profundo	Implementada e validada funcionalmente	Usa modos de baixo consumo entre contactos quando a válvula simulada está fechada, com a mesma troca de mensagens LoRa.
LPC1769 com sono profundo	Implementada e validada funcionalmente	Usa sono profundo com despertar por RTC entre contactos, com a mesma troca de mensagens LoRa.

A Figura 4.3 mostra o consumo diário estimado para as quatro variantes, usando a mesma carga de referência: dois ciclos de rega por dia e os mesmos parâmetros LoRa. Nas versões com WFI, a energia de base da placa e do rádio em *standby* domina o total diário. Por isso, aumentar o intervalo de *check-in* reduz pouco a estimativa: no STM32L432KC passa de cerca de 1219 para 1082 mWh/dia entre 10 s e 1 h; no LPC1769 passa de cerca de 4014 para 3877 mWh/dia. Nas variantes com sono profundo, o intervalo passa a ser o fator dominante no modelo: o STM32L432KC passa de cerca de 65,4 mWh/dia a 30 s para cerca de 7,0 mWh/dia a 15 min, e o LPC1769 passa de cerca de 115,1 mWh/dia a 30 s para cerca de 8,6 mWh/dia a 15 min. As variantes com sono profundo foram validadas funcionalmente com o mesmo protocolo, formato de pacotes e configuração LoRa. A poupança energética apresentada é uma estimativa de modelo e deve ser confirmada por medição elétrica no hardware final.

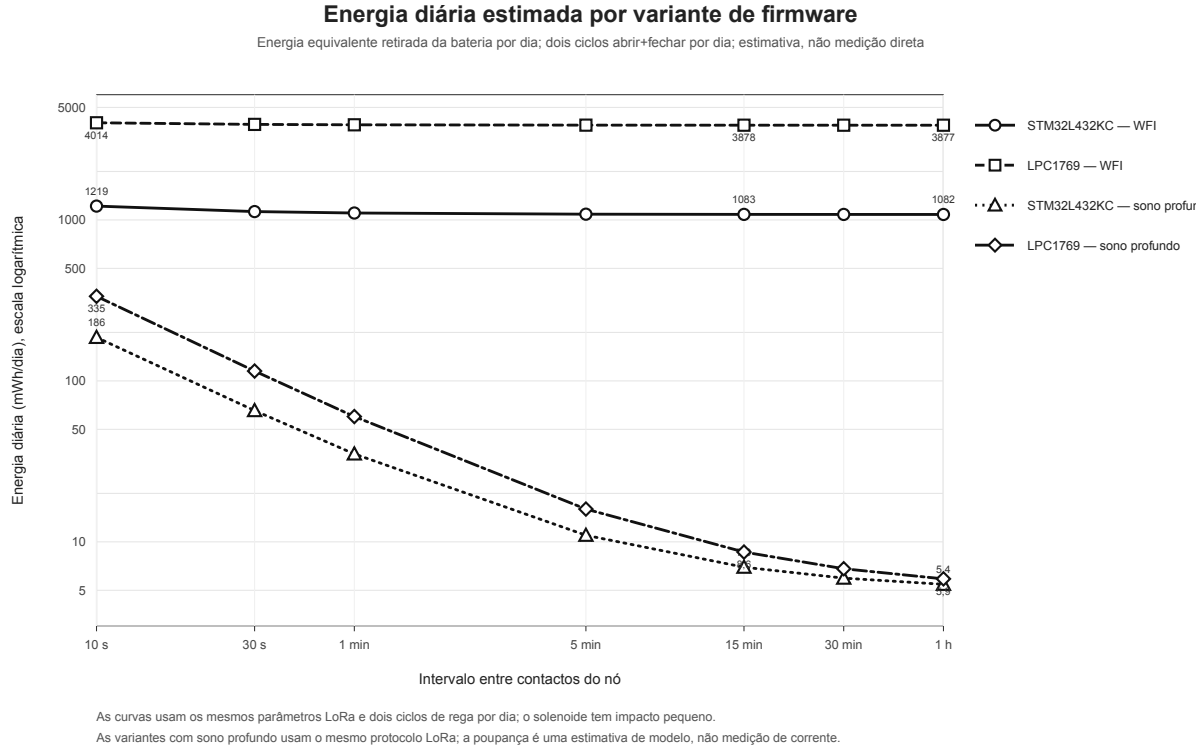


Figura 4.3: Estimativa de energia diária retirada da bateria em função do intervalo de *check-in*. As curvas comparam as versões STM32L432KC com WFI, LPC1769 com WFI, STM32L432KC com sono profundo e LPC1769 com sono profundo, todas com os mesmos parâmetros LoRa e dois ciclos de rega por dia.

A Tabela 4.6 resume alguns pontos da mesma estimativa. A potência média é apenas a energia diária

dividida por 24 h; a autonomia real deve ser medida com a bateria, regulador, sensores e driver finais.

Tabela 4.6: Pontos representativos da estimativa de energia diária

Variante	**Intervalo**	**Ciclos/dia**	**Energia diária (mWh/dia)**	**Potência média (mW)**
STM32L432KC com WFI	10 s	2	1219,0	50,8
STM32L432KC com WFI	1 h	2	1081,5	45,1
LPC1769 com WFI	10 s	2	4014,3	167,3
LPC1769 com WFI	1 h	2	3876,8	161,5
STM32L432KC com sono profundo	30 s	2	65,4	2,73
STM32L432KC com sono profundo	15 min	2	7,0	0,29
LPC1769 com sono profundo	30 s	2	115,1	4,80
LPC1769 com sono profundo	15 min	2	8,6	0,36

4.7 Perfil de *Runtime* da Implementação

Para caracterizar o comportamento interno do firmware, foram gerados perfis de *runtime* a partir de amostras recolhidas com OpenOCD/GDB e renderizados com a ferramenta FlameGraph. A Tabela 4.7 resume os dados recuperados dos arquivos de captura.

Tabela 4.7: Metodologia das capturas usadas nos perfis de *runtime*

Perfil	**Duração aprox.**	**Instantes**	**Observações representadas**	**Taxa efetiva**
ESP32-S3 HTTP/LoRa	74 s	119	1 645 pilhas de tarefas	1,61 Hz
STM32L432KC WFI	205 s	779	779 amostras	3,80 Hz
STM32L432KC sono profundo	293 s	450	155 amostras ativas	1,54 Hz
LPC1769, janela ativa	229 s	1 200	23 amostras ativas	5,24 Hz

Em cada instante, o GDB retoma e suspende o alvo para recolher a pilha de chamadas; por isso, a captura perturba a execução e não constitui uma medição temporal não intrusiva. No ESP32-S3 são recolhidas várias pilhas de tarefas por suspensão. Nos perfis ativos do STM32L432KC e do LPC1769, as amostras de espera foram removidas; o perfil LPC1769 deve ser interpretado com especial cautela por conservar apenas 23 amostras ativas. A largura de cada *frame* representa contagens de amostras, não tempo de execução nem consumo energético.

A Figura 4.4 apresenta o perfil do concentrador ESP32-S3 durante uma carga com pedidos HTTP e atividade LoRa. O gráfico mostra a coexistência dos serviços de rede, API HTTP local e rádio no firmware do concentrador.

No concentrador, os ramos mais visíveis correspondem às tarefas `httpd`, `tcpip` e `radio_service`. As pilhas associadas a `handle_node_status`, `handle_decoded_packet` e `send_packet` representam o caminho entre a receção de uma mensagem LoRa, o tratamento aplicacional e a preparação da resposta. Os ramos de HTTP correspondem à carga gerada pela interface Web.

A Figura 4.5 apresenta o controlador STM32L432KC na variante WFI. Nesta variante, o processador entra em espera por interrupção quando o FreeRTOS não tem trabalho ativo.

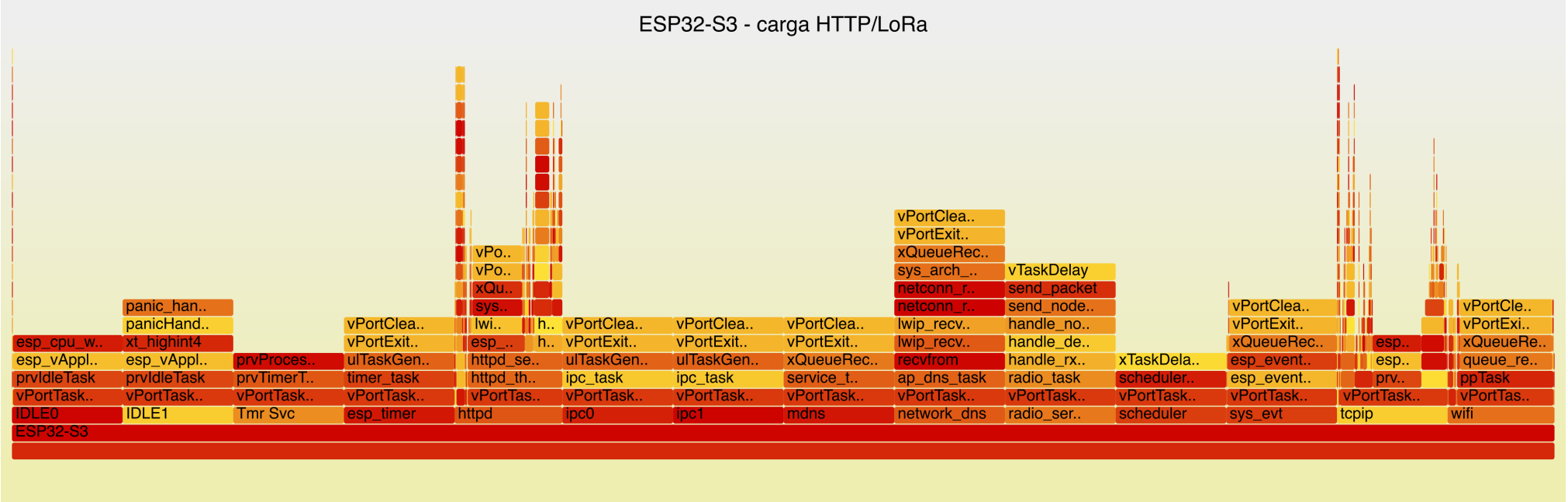


Figura 4.4: *Flame graph* do concentrador ESP32-S3 durante carga HTTP/LoRa.

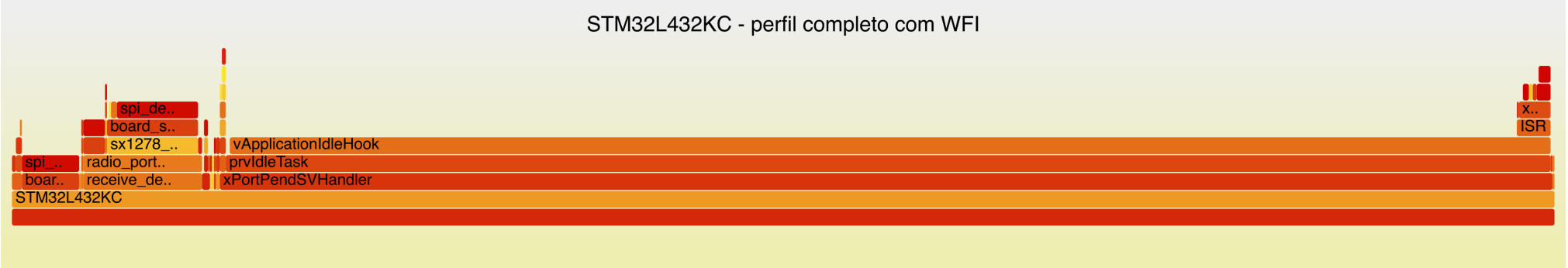


Figura 4.5: *Flame graph* do controlador STM32L432KC na variante WFI.

A largura dominante do ramo `prvIdleTask` → `vApplicationIdleHook()` mostra que a maioria das amostras ocorre em estado de espera. Os ramos menores, como `radio_port_receive`, `receive_decoded_packet`, `sx1278_read_reg` e `board_spi_transfer`, correspondem aos períodos em que o controlador acorda para trocar mensagens por LoRa.

A Figura 4.6 mostra o mesmo controlador na variante de sono profundo. Como a suspensão não aparece como trabalho de CPU, o gráfico representa apenas a janela ativa após despertar.

Neste caso, as amostras concentram-se no caminho de execução `node_runtime_run`, `node_app_task`, `wait_for_response_after_status` e `receive_decoded_packet`. A presença de `radio_port_receive`, `sx1278_read_reg` e `board_spi_transfer` mostra que a janela ativa é dominada pela receção LoRa e pelo acesso SPI ao módulo RA-02/SX1278.

A Figura 4.7 apresenta o perfil do controlador LPC1769 durante uma janela ativa do firmware.

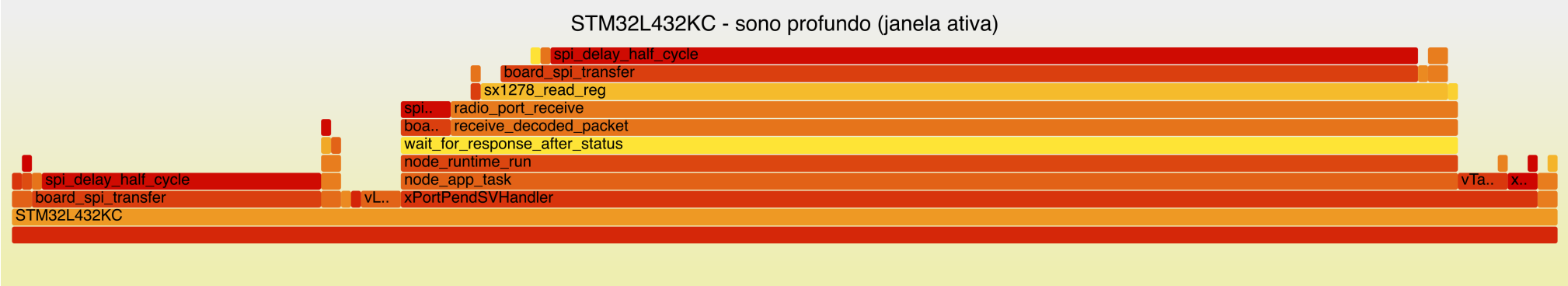


Figura 4.6: *Flame graph* do controlador STM32L432KC na variante de sono profundo, limitado à janela ativa após despertar.

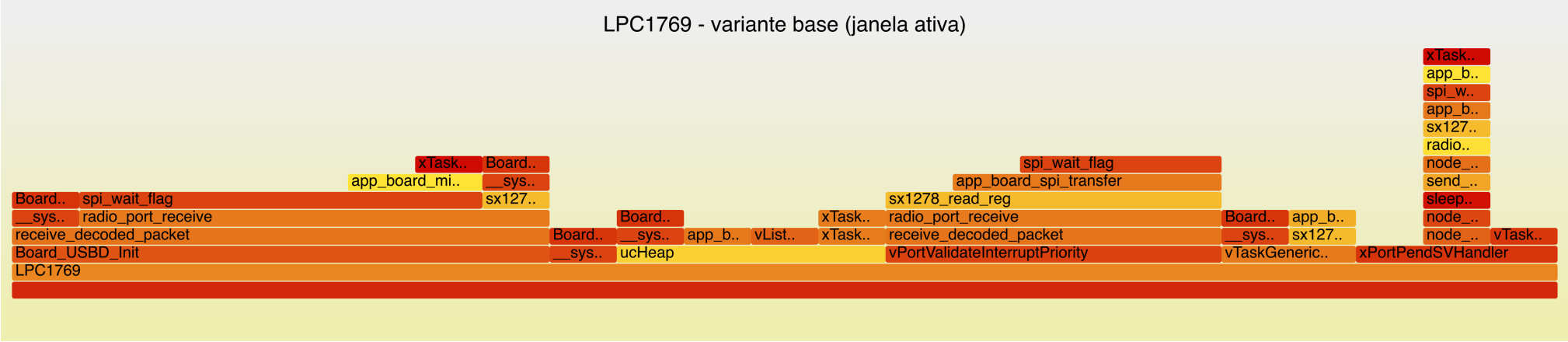


Figura 4.7: *Flame graph* do controlador LPC1769 durante uma janela ativa do firmware.

No LPC1769 observa-se uma estrutura semelhante à do STM32L432KC durante a atividade rádio: receção de pacotes, descodificação, acesso ao transcetor por SSP/SPI e chamadas do FreeRTOS associadas ao escalonamento. Isto confirma que a lógica portátil do controlador segue o mesmo fluxo, ficando as diferenças principais concentradas na camada específica da placa.

4.8 API HTTP e Interface Web

O componente `http_api` usa o servidor HTTP embebido do ESP-IDF e a biblioteca `cJSON` para interpretar pedidos e gerar respostas JSON. A interface Web é servida localmente pelo próprio concentrador e comunica apenas com as rotas da API HTTP em `/api`.

Os grupos de rotas implementados abrangem:

- estado do sistema, tempo, rede e diagnóstico de rádio;
- abertura, consulta e fecho da janela de emparelhamento;
- listagem, detalhe, renomeação e remoção de controladores conhecidos;
- criação de comandos de abertura ou fecho da saída simulada;
- listagem, criação, atualização e remoção de horários;
- consulta dos eventos recentes mantidos em memória.

A interface Web permite executar localmente as operações essenciais do protótipo: observar controladores, abrir a janela de emparelhamento, nomear válvulas, pedir abertura ou fecho, configurar horários, ajustar a hora manualmente e configurar credenciais Wi-Fi STA mantendo o ponto de acesso de recuperação ativo.

Tabela 4.8: Exemplos de rotas HTTP usadas pela interface Web

Rota	**Método**	**Função**
`/api/system/status`	GET	Estado geral do sistema e contadores de HTTP.
`/api/radio/status`	GET	Estado do serviço LoRa, RSSI, SNR e contadores RX/TX.
`/api/nodes`	GET	Lista de controladores conhecidos.
`/api/nodes/<id>/commands`	POST	Criação de comando `open_valve` ou `close_valve`.
`/api/schedules`	GET/POST	Consulta e criação de horários locais.
`/api/events/recent`	GET	Eventos recentes mantidos em memória.

4.9 Síntese

A implementação atual integra concentrador, controladores FreeRTOS, protocolo partilhado, comunicação LoRa real, persistência local e interface de controlo Web. O resultado é uma base funcional para validação experimental do sistema de rega, com a lógica principal concentrada no `app_core` e com separação clara entre domínio, transporte rádio, persistência, interface Web e API HTTP.

5 Validação e Testes

5.1 Metodologia de Validação

A validação do sistema combinou verificações de compilação, testes funcionais com hardware real e recolha de métricas através da API HTTP local do concentrador. O objetivo foi confirmar o ciclo essencial do sistema: arranque do concentrador ESP32-S3, comunicação LoRa com controladores FreeRTOS, emparelhamento, verificações periódicas, entrega de comandos, confirmação por `COMMAND_ACK` e observabilidade por HTTP/JSON.

Foram consideradas quatro frentes de validação:

- **verificações de software**: compilação do firmware do concentrador em ESP-IDF, incluindo armazenamento, rede, tempo, núcleo aplicacional, HTTP, codec LoRa, serviço de rádio e controlador SX1278;
- **verificações dos controladores**: compilação dos firmwares STM32L432KC e LPC1769, juntamente com os módulos comuns de `node-firmware/common`, e execução dos testes unitários da persistência comum;
- **testes funcionais de hardware**: arranque do concentrador e dos controladores, tráfego real entre módulos RA-02/SX1278, emparelhamento, envio de estado, entrega de comandos e confirmação por `COMMAND_ACK`;
- **recolha de métricas**: amostragem de estado, contadores de rádio, RSSI, SNR, intervalos de verificação, latência de comandos e tempo de resposta da API HTTP local.

Os testes foram executados com o concentrador acessível em rede local e com dois controladores emparelhados em simultâneo. O controlador 1 correspondeu à placa STM32L432KC e o controlador 2 à placa LPC1769. A validação quantitativa incluiu recolhas curtas, testes de comando, testes de alcance sem linha de vista e uma execução contínua de aproximadamente uma hora para avaliação de estabilidade, sempre com amostragem pelas rotas `/api/system/status`, `/api/radio/status` e `/api/nodes`. A escala experimental ficou, portanto, limitada a dois controladores reais ligados em simultâneo.

5.2 Testes Realizados

A Tabela 5.1 resume os principais testes usados para sustentar a validação experimental do protótipo integrado. Estes testes cobrem desde a compilação e arranque dos firmwares até ao emparelhamento, comunicação LoRa, execução de comandos, persistência local e utilização da interface Web. O objetivo não foi apenas verificar componentes isolados, mas confirmar que o ciclo completo de comando, confirmação e atualização da saída simulada funciona de forma coerente entre o concentrador, os controladores e a interface Web local.

Tabela 5.1: Testes realizados no protótipo integrado

Teste	**Observações**
Compilação e arranque	Firmware do concentrador e dos dois controladores construído com sucesso; serviços principais inicializados sem erro.
Emparelhamento LoRa	Dois controladores emparelhados, com resposta ao pedido de emparelhamento e registo concluído no primeiro *check-in*.
Comunicação controlador–concentrador	Receção periódica de estado dos controladores e atualização dos dados disponíveis na interface Web.
Operação da saída de atuação simulada	Comandos de abertura e fecho entregues ao controlador, refletidos no LED integrado da placa e confirmados por uma mensagem de confirmação (ACK).
Persistência dos controladores	Quatro testes unitários concluídos com sucesso: gravação e restauro, deduplicação após perda do ACK e reinicialização, fecho seguro persistido e rejeição de registos inválidos.
Tempo de resposta da API HTTP	Cem pedidos consecutivos na rede local, com dois controladores registados; todas as respostas HTTP 200 e latência máxima inferior a 1 s.
Testes de alcance sem linha de vista	Varredura de distâncias com obstáculos, registando RSSI, SNR, contadores RX/TX e latência de comandos; os resultados servem para caracterizar o ambiente testado, não o alcance máximo absoluto.
Interface, horários e estado local	Consulta de estado, eventos e controladores através da interface Web, incluindo criação/listagem de horários.
Teste contínuo de aproximadamente uma hora	Execução contínua de cerca de uma hora com dois controladores ativos, sem novos erros de descodificação, transmissão ou núcleo aplicacional.

5.3 Validação do Protocolo LoRa

A validação da comunicação rádio confirmou o fluxo implementado entre concentrador e controladores. O emparelhamento foi iniciado por `PAIR_REQUEST`, a resposta do concentrador foi associada ao pedido correto e o registo ficou concluído com o primeiro `NODE_STATUS` válido.

Nos testes de abertura e fecho, cada controlador aplicou o comando recebido à respetiva saída simulada, transmitiu `COMMAND_ACK` e voltou a reportar estado através de `NODE_STATUS`. Este ciclo confirmou a consistência entre protocolo, firmware dos controladores e núcleo aplicacional do concentrador.

A persistência comum foi verificada em ambiente *host* com quatro testes automatizados: gravação e restauro do estado, reinicialização após perda do ACK sem reaplicação do comando, persistência do fecho seguro e rejeição de registos inválidos ou desatualizados. Todos os testes terminaram com sucesso.

5.4 Resultados de Comunicação

Durante a recolha de 300 s foram observados dois controladores ativos. A Tabela 5.2 resume as métricas principais do teste. Os valores de RSSI e SNR foram obtidos a partir do último pacote recebido pelo concentrador em cada amostra.

Tabela 5.2: Métricas principais de comunicação no teste de 300 s

Métrica	**Valor observado**
Duração da recolha	300 s
Amostras da API	150
Pacotes RX	54
Pacotes descodificados	54
Erros de descodificação	0
Pacotes TX	54
Erros TX	0
RSSI mínimo / médio / máximo	-80 / -63,99 / -55 dBm
SNR mínimo / médio / máximo	7,0 / 10,91 / 13,0 dB

Os valores da Tabela 5.2 são suficientes para caracterizar esta recolha: os 54 pacotes recebidos foram descodificados e respondidos sem novos erros de RX ou TX, e os valores de RSSI/SNR mantiveram margem positiva para o cenário de testes.

5.5 Testes de Alcance e Qualidade da Comunicação

Foram realizados testes adicionais para caracterizar a comunicação LoRa em ambiente prático sem linha de vista. O concentrador ESP32-S3 permaneceu fixo junto do computador de teste e os controladores foram deslocados para distâncias aproximadas. Os resultados apresentados nas figuras correspondem a um ensaio residencial/urbano, com bom tempo e sem linha de vista direta. O ambiente tinha obstáculos e não foi preparado como ensaio exterior normalizado; por isso, os resultados devem ser lidos como a capacidade observada neste cenário específico, não como alcance máximo absoluto.

A configuração rádio foi mantida constante nos testes: módulos RA-02/SX1278 a 433,920 MHz, largura de banda de 125 kHz, SF9, codificação 4/5, preâmbulo de 8 símbolos, *sync word* 0x12, CRC ativo e potência de transmissão de aproximadamente +17 dBm. A comunicação usada foi LoRa ponto-a-ponto, não LoRaWAN.

Em cada ponto foi feita uma captura por API, seguida, quando existia comunicação, por seis comandos `close_valve`. Foram registados os deltas dos contadores RX, pacotes descodificados, TX, erros de descodificação, erros de aplicação e erros TX, bem como RSSI, SNR e latência dos comandos. Nos pontos sem novos pacotes recebidos, os comandos foram omitidos e os valores RSSI/SNR expostos pela API foram ignorados, pois correspondiam ao último pacote recebido anteriormente.

A Figura 5.1 apresenta os valores de RSSI e SNR obtidos nos testes de alcance. Como esperado num ambiente sem linha de vista, os valores não são estritamente monótonos com a distância, devido a obstruções, reflexões, orientação das antenas e posição relativa dos controladores. Ainda assim, observa-se a redução gradual da margem de rádio: nos pontos longos, os valores médios ficaram próximos de -130 dBm ou inferiores, com SNR frequentemente negativo.

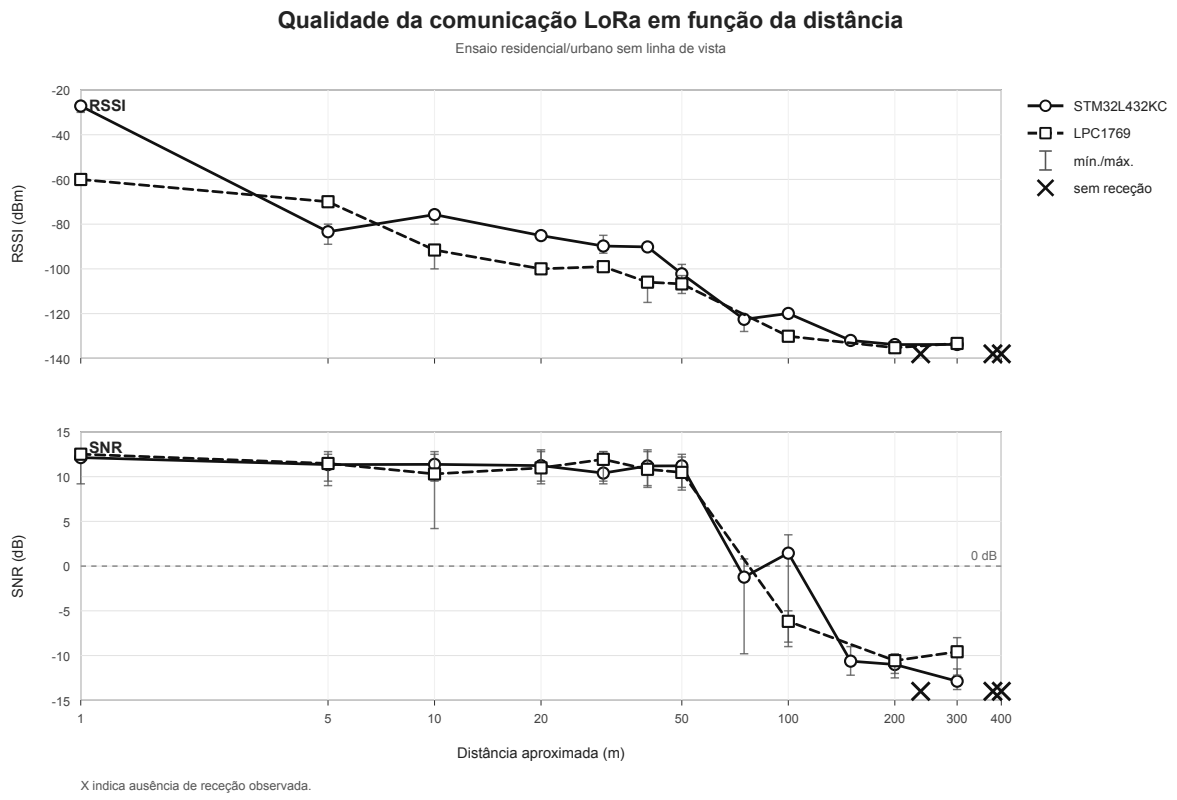


Figura 5.1: Qualidade da comunicação LoRa nos testes de alcance em ambiente residencial/urbano sem linha de vista.

A Figura 5.2 relaciona a taxa de pacotes recebidos com a latência dos comandos. Até 50 m, ambos os controladores mantiveram receção regular e todos os comandos tentados foram concluídos. No STM32L432KC, os pontos até 200 m mantiveram contadores RX/descodificados/TX coerentes e comandos concluídos em 6/6 testes, embora os pontos longos já apresentassem margem de rádio fraca. A 300 m ainda houve pacotes e os seis comandos concluíram, mas a captura tornou-se intermitente, com apenas 6 pacotes recebidos em 180 s e latência máxima de 88,9 s. A 400 m não foi observada receção para o STM32L432KC.

No LPC1769, a operação foi regular até 50 m. O ponto a 100 m funcionou com margem já fraca, e os pontos a 200 m e 300 m também concluíram todos os comandos com margem de rádio muito reduzida: a 300 m, por exemplo, o RSSI médio foi cerca de -133,3 dBm e o SNR médio cerca de -9,6 dB. A 400 m não foi observada receção durante 180 s.

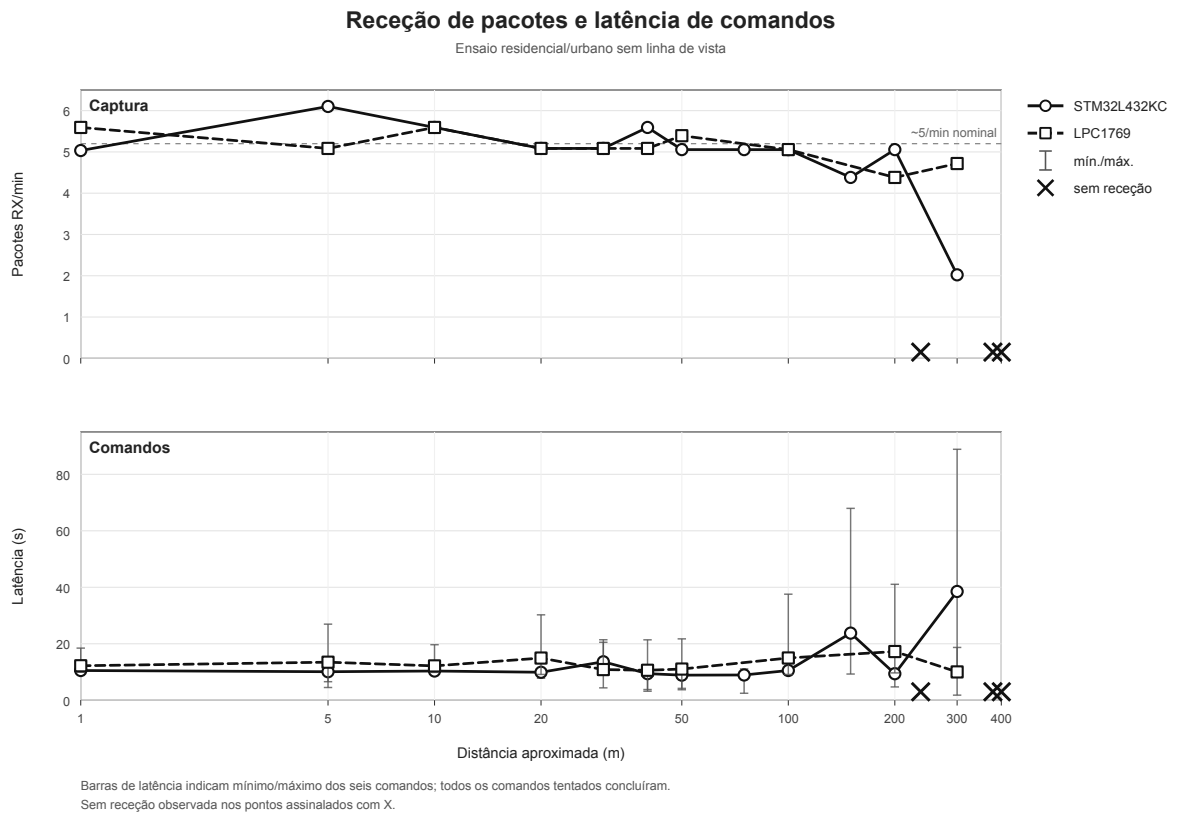


Figura 5.2: Receção de pacotes e latência de comandos durante os testes de alcance.

Estes resultados validam o funcionamento do protótipo nas distâncias e condições testadas, mas não constituem validação de alcance de 1–2 km nem de alcance máximo. Um alcance exterior superior é plausível em LoRa/SF9 com antenas, altura e condições favoráveis, mas deve ser confirmado por ensaios próprios em linha de vista ou em cenários exteriores com maior controlo.

5.6 Teste Contínuo de Uma Hora

Além das recolhas curtas, foi realizada uma execução contínua de cerca de uma hora com o concentrador e dois controladores emparelhados. Durante esse período, o concentrador registou 598 pacotes recebidos e descodificados, 598 respostas transmitidas e zero erros de descodificação, transmissão ou núcleo aplicacional. As falhas observadas limitaram-se a leituras HTTP falhadas entre o computador de teste e a API local, não a erros LoRa nem a falhas internas do sistema.

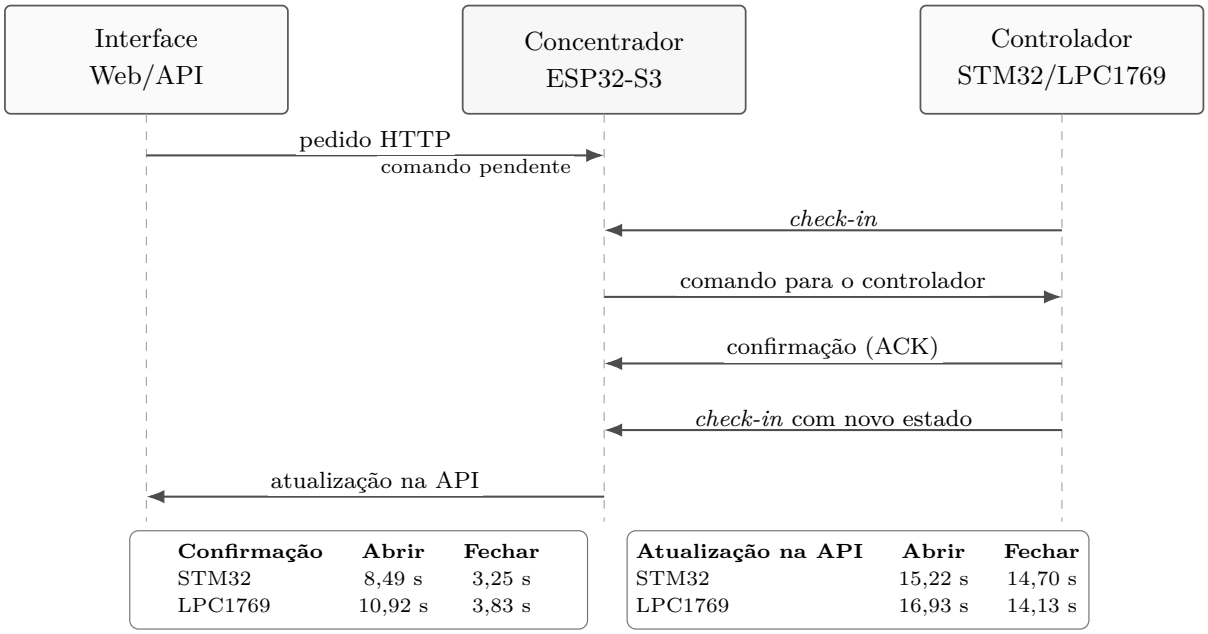
5.7 Intervalos de Verificação

Os dois controladores foram configurados para verificações frequentes, adequadas aos testes. Na recolha inicial de 300 s, a média ficou próxima de 11,6 s para os dois controladores, resultado do intervalo configurado e do *jitter* introduzido para evitar sincronização permanente. Nos dois testes, ambos mantiveram intervalos médios de contacto próximos de 12 s: cerca de 11,58 s na recolha de 300 s e entre 12,00 s e 12,12 s na execução contínua de aproximadamente uma hora. Alguns intervalos máximos foram superiores devido ao espaçamento natural entre contactos e a pequenas variações de temporização.

5.8 Latência de Comandos

A entrega de comandos ocorre quando o controlador faz o contacto seguinte com o concentrador. Por isso, a latência observada é dominada pelo intervalo de verificação do controlador e pelo instante em que o pedido HTTP é criado dentro desse intervalo. Foram realizados dois tipos de teste: um ciclo funcional de abertura e fecho em cada controlador e um teste repetido de comandos de fecho.

No ciclo funcional, ambos os controladores aceitaram os comandos, limparam o comando pendente após confirmação e atualizaram posteriormente o estado lógico reportado da saída simulada. A Figura 5.3 ilustra a diferença entre a latência até à confirmação do comando e a latência até ao estado esperado ser visível na API. A segunda é naturalmente maior porque depende de um novo *check-in* do controlador após a execução.



Confirmação	Abrir	Fechar
STM32	8,49 s	3,25 s
LPC1769	10,92 s	3,83 s

Atualização na API	Abrir	Fechar
STM32	15,22 s	14,70 s
LPC1769	16,93 s	14,13 s

Figura 5.3: Ciclo observado entre pedido de comando, confirmação e atualização na API.

Os valores indicados na figura correspondem aos intervalos observados nos testes realizados.

Para avaliar repetibilidade sem reabrir as saídas, foram ainda enviados 10 comandos `close_valve`, cinco para cada controlador. Todos os comandos foram aceites pela API e concluídos sem *timeout*, mantendo as saídas simuladas no estado lógico fechado. As latências de confirmação ficaram entre 7,36 s e 12,87 s, com média global de 10,78 s.

O mesmo identificador de pedido foi repetido de forma imediata e a API devolveu a mesma resposta lógica. Isto evita a criação acidental de comandos duplicados quando o cliente repete um pedido por falha de rede.

5.9 Tempo de Resposta da API HTTP

Para validar o RNF-03, foram efetuados 100 pedidos GET consecutivos a partir de um único cliente na rede local, com os dois controladores registados e *online*. Após cinco pedidos de preparação excluídos dos resultados, foram enviados 25 pedidos a cada uma das rotas `/api/system/status`, `/api/radio/status`, `/api/nodes` e `/api/events/recent`, de forma sequencial e com uma nova ligação TCP em cada pedido. Todas as respostas apresentaram estado HTTP 200.

A Tabela 5.3 apresenta os resultados globais. Para calcular os percentis, os 100 tempos de resposta foram ordenados do menor para o maior.

Tabela 5.3: Tempo de resposta da API HTTP em 100 pedidos locais

Métrica	**Tempo de resposta**
Mediana	21,10 ms
P95	48,08 ms
P99	49,14 ms
Máximo	115,54 ms

A mediana representa o centro da amostra. O P95 corresponde ao 95.º valor da lista ordenada: 95 pedidos responderam em até 48,08 ms e cinco demoraram mais. O P99 corresponde ao 99.º valor: 99 pedidos responderam em até 49,14 ms, ficando apenas um pedido acima desse valor. Esse pedido foi também o mais lento, com 115,54 ms, abaixo do limite de 1 s definido no RNF-03.

5.10 Estimativa Energética e Limites da Medição

Não foi realizado teste de autonomia em bateria nem medição direta de corrente com analisador externo. A avaliação energética apresentada neste relatório é, por isso, uma estimativa de dimensionamento baseada em ciclo de trabalho e em valores de referência dos componentes, conforme apresentado na Tabela 4.5 e na Figura 4.3.

As variantes com sono profundo mantêm o mesmo protocolo, os mesmos formatos de pacote e os mesmos parâmetros RA-02/SX1278 das versões com WFI. Assim, os testes de alcance caracterizam a configuração rádio comum. A validação dessas variantes confirmou arranque, contacto periódico, receção de comandos e recuperação após sono profundo; a autonomia real deve ser medida no hardware completo, incluindo bateria, regulador, sensores e driver de solenoide.

5.11 Colisões, Repetições e Temporização

O protótipo não implementa deteção completa de portadora, CAD ou acesso determinístico por TDMA. A mitigação de contenção assenta em tempos de espera, expiração de respostas e *jitter*/*backoff* aleatório limitado nos controladores, tanto em novas tentativas de emparelhamento como em verificações periódicas.

Esta solução é adequada para a escala validada do protótipo, com número reduzido de controladores e tráfego pouco frequente. Contudo, não substitui uma análise detalhada de capacidade do canal nem uma política de acesso coordenado para redes com muitos controladores ou tráfego concentrado.

5.12 Limitações da Validação

Apesar de o ciclo funcional principal ter sido validado com hardware real, a validação realizada mantém limites claros:

- os testes de alcance sem linha de vista não substituem ensaios de alcance, perdas e robustez em ambiente exterior com maior controlo ou em linha de vista;
- os testes com múltiplos controladores em atividade simultânea foram limitados a dois controladores reais;
- a segurança da interface Web, da API HTTP e do protocolo rádio não foi objetivo principal desta fase de validação;
- a válvula real foi substituída pelos LEDs integrados das placas, sem driver de solenoide integrado.

5.13 Síntese

A validação efetuada confirma o funcionamento integrado do sistema: o concentrador arranca, expõe controlo local por API/Web, emparelha controladores FreeRTOS por LoRa, recebe estados, entrega comandos de abertura ou fecho e processa confirmações. Os resultados de comunicação foram estáveis tanto na recolha inicial de 300 s como na execução contínua de aproximadamente uma hora, sem novos erros de descodificação, transmissão ou núcleo aplicacional. Os testes de alcance sem linha de vista mostraram a degradação esperada de RSSI/SNR com obstáculos: o STM32L432KC manteve comunicações completas até 200 m, já com margem fraca nos pontos longos, e comportamento intermitente a 300 m, enquanto o LPC1769 funcionou até 300 m com margem fraca e sem receção observada a 400 m. Estes resultados não validam alcance de quilómetros. As estimativas de consumo mostram que as variantes com sono profundo são a direção mais promissora para reduzir a energia diária entre contactos, mas a autonomia final requer medição elétrica no hardware completo.

6 Conclusões e Trabalho Futuro

6.1 Conclusões

O trabalho desenvolvido resultou num protótipo funcional de sistema de rega inteligente com concentrador ESP32-S3, controladores físicos STM32L432KC e LPC1769 e comunicação LoRa ponto-a-ponto sobre módulos RA-02/SX1278. A solução demonstra o ciclo essencial de operação entre concentrador, controladores FreeRTOS, interface Web e API HTTP.

Foram alcançados os principais objetivos definidos para o protótipo:

- implementação do concentrador em ESP-IDF, com serviços de armazenamento, rede, tempo, núcleo aplicacional, HTTP e rádio;
- implementação de firmware de controlador FreeRTOS para STM32L432KC e LPC1769, apoiado por módulos C portáveis em `node-firmware/common`;
- comunicação LoRa real entre concentrador e controladores, incluindo emparelhamento com correlação do pedido e registo concluído no primeiro `NODE_STATUS` válido;
- protocolo binário compacto de tamanho fixo, sem versão, comprimento, checksum aplicacional, hora do servidor ou versão de configuração na comunicação rádio;
- entrega de comandos `RESP_OPEN_VALVE` e `RESP_CLOSE_VALVE`, com confirmação por `COMMAND_ACK`;
- interface Web local e API HTTP em `/api` para observação do sistema, gestão de controladores, comandos, horários e eventos recentes;
- persistência local de configurações e do estado aplicacional do concentrador, recorrendo a NVS e LittleFS, e persistência de identidade, intervalos, estado lógico e último resultado de comando nos controladores.

A arquitetura adotada mostrou-se adequada ao objetivo do projeto: cada controlador corresponde a uma válvula, o concentrador mantém um único comando em espera por controlador, as respostas rádio são tipos fixos e uma resposta sem comando não transporta campos de comando. Esta simplificação reduziu a complexidade do firmware do controlador, diminuiu o tempo no ar dos pacotes e facilitou a validação de ponta a ponta.

A validação final confirmou a operação simultânea dos dois controladores reais durante cerca de uma hora, com contactos periódicos, telemetria e ciclos de abertura e fecho refletidos nos LEDs integrados das placas, sem falhas observadas de protocolo ou rádio.

Nos testes de alcance sem linha de vista, observou-se comunicação até 300 m, embora com margem reduzida e receção intermitente na maior distância. Este resultado caracteriza o cenário ensaiado, mas não valida alcance quilométrico.

A análise energética reforçou duas conclusões de arquitetura. Primeiro, as versões base em placas de desenvolvimento continuam dominadas pelo consumo de *standby* da placa e do rádio, pelo que o intervalo de *check-in* tem impacto limitado sem sono profundo. As variantes STM32L432KC e LPC1769 com sono profundo, validadas funcionalmente mantendo o mesmo protocolo e a mesma configuração LoRa, mostram no modelo uma redução significativa do consumo diário estimado. Segundo, usando uma válvula biestável de 9 V, a energia de atuação por pulso é pequena face à energia diária do rádio e da eletrónica

auxiliar; o desafio principal do atuador é fornecer o pico de corrente e a inversão de polaridade com segurança, não a energia média da abertura/fecho.

6.2 Limitações Identificadas

Apesar do funcionamento integrado validado, o protótipo não deve ser interpretado como produto final. As principais limitações identificadas são:

- autonomia e consumo ainda avaliados por modelo, sem teste em bateria nem medição direta da corrente nos diferentes estados do sistema completo;
- ensaios de alcance e robustez ainda limitados ao cenário testado, sem validação em linha de vista, terrenos extensos ou instalações com diferentes antenas;
- testes com múltiplos controladores em atividade simultânea limitados a dois controladores reais;
- segurança local e rádio ainda não preparada para instalação real;
- ausência de integração da válvula hidráulica e do respetivo driver de potência; a atuação foi representada pelos LEDs integrados das placas, sem bateria, proteção e caixa finais;

6.3 Trabalho Futuro

As próximas etapas podem seguir duas direções complementares: tornar o sistema de rega mais robusto para utilização real e adaptar a mesma arquitetura a outros cenários de atuação remota. Destacam-se:

- medir autonomia em cenários de longa duração, definir métodos de medição de corrente para sono profundo, receção, transmissão e atuação da válvula, e integrar a válvula final com ponte H ou driver equivalente, geração da tensão necessária ao solenoide, proteção indutiva, caixa, conectores e proteção ambiental;
- estudar diferentes combinações de bateria, driver e antena LoRa para casos de uso distintos, incluindo instalações residenciais compactas e explorações agrícolas com bateria de maior capacidade, alimentação solar, driver dimensionado para válvulas ou atuadores mais exigentes e antena com melhor posicionamento ou ganho;
- otimizar o firmware, reduzindo tempo acordado, acessos ao rádio, consumo em espera, tamanho das mensagens e trabalho repetido nas tarefas FreeRTOS, incluindo melhorias nos modos de baixo consumo e nas janelas de receção;
- melhorar a interface Web com funcionalidades de operação, como impedir ou permitir válvulas em simultâneo, suspender grupos de horários, alterar rapidamente a duração de rega através de um fator de ajuste (*water budget*) e apresentar diagnóstico mais claro para instalação e manutenção;
- reforçar autenticação, autorização, proteção da interface Web e da API HTTP, segurança do emparelhamento, integridade das mensagens rádio e gestão de chaves, uma vez que esta área não foi aprofundada no protótipo atual;
- explorar a adaptação da arquitetura a outros atuadores. Alterando o driver, a alimentação e a camada mecânica, o mesmo princípio de concentrador, controlador remoto e comando confirmado pode ser aplicado a bombas, relés, portões ou outros equipamentos que apenas necessitem de ações de abertura, fecho, ligar ou desligar.

Referências

[1] Rain Bird Corporation, "Rain Bird smart irrigation solutions," consultado em: 9 abr. 2026. [Online]. Available: https://store.rainbird.com/st8-2-0-controllers-8-zone-smart-irrigation-wifi.html

[2] Hunter Industries, "Hydrawise smart irrigation control," consultado em: 9 abr. 2026. [Online]. Available: https://www.hydrawise.com/

[3] Rachio, "Rachio smart watering devices," consultado em: 9 abr. 2026. [Online]. Available: https://rachio.com/

[4] Orbit Irrigation, "Orbit B-hyve smart watering products," consultado em: 9 abr. 2026. [Online]. Available: https://www.orbitonline.com/collections/b-hyve

[5] Ecowitt, "Ecowitt WFC01 smart hose water timer," consultado em: 9 abr. 2026. [Online]. Available: https://shop.ecowitt.com/en-gb/products/wfc01

[6] Shelly, "Shelly Pro LoRa Add-on," consultado em: 21 jun. 2026. [Online]. Available: https://www.shelly.com/products/shelly-pro-lora-add-on

[7] OpenSprinkler, "OpenSprinkler," consultado em: 9 abr. 2026. [Online]. Available: https://opensprinkler.com/

[8] D. Vallejo-Gómez, M. Osorio, and C. A. Hincapié, "Smart irrigation systems in agriculture: A systematic review," *Agronomy*, vol. 13, no. 2, p. 342, 2023.

[9] IEEE, "IEEE standard for low-rate wireless networks (IEEE 802.15.4-2020)," IEEE Std 802.15.4-2020, consultado em: 9 abr. 2026. [Online]. Available: https://standards.ieee.org/ieee/802.15.4/7029/

[10] Connectivity Standards Alliance, "Zigbee specification," consultado em: 9 abr. 2026. [Online]. Available: https://csa-iot.org/all-solutions/zigbee/

[11] 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.

[12] LoRa Alliance, "LoRaWAN L2 1.0.4 specification," consultado em: 9 abr. 2026. [Online]. Available: https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification

[13] J. Ding, M. Nemati, C. Ranaweera, and J. Choi, "IoT connectivity technologies and applications: A survey," *IEEE Access*, vol. 8, pp. 67 646–67 673, 2020.

[14] Connectivity Standards Alliance, "Zigbee frequently asked questions," consultado em: 10 jul. 2026. [Online]. Available: https://csa-iot.org/all-solutions/zigbee/zigbee-faq/

[15] Thread Group, "Thread benefits," consultado em: 10 jul. 2026. [Online]. Available: https://threadgroup.org/What-is-Thread/Thread-Benefits

[16] Semtech Corporation, “What are LoRa and LoRaWAN?” consultado em: 10 jul. 2026. [Online]. Available: https://www.semtech.com/lora/what-is-lora

[17] LoRa Alliance, “What is LoRaWAN?” consultado em: 10 jul. 2026. [Online]. Available: https://lora-alliance.org/about-lorawan/

[18] Bluetooth SIG, “Understanding bluetooth range,” consultado em: 10 jul. 2026. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/

[19] ——, “The fundamental concepts of bluetooth mesh networking,” consultado em: 10 jul. 2026. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/mesh/

[20] *STM32WLE5xx STM32WLE4xx Datasheet*, STMicroelectronics, consultado em: 9 abr. 2026. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32wle5jc.pdf

[21] Seeed Studio, “Wio-E5 wireless module,” consultado em: 10 jul. 2026. [Online]. Available: https://wiki.seeedstudio.com/LoRa-E5_STM32WLE5JC_Module/

[22] RAKwireless, “RAK3172 wisduo module overview,” consultado em: 10 jul. 2026. [Online]. Available: https://docs.rakwireless.com/product-categories/wisduo/rak3172-module/overview/

[23] STMicroelectronics, “NUCLEO-WL55JC development board,” consultado em: 9 abr. 2026. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-wl55jc.html

[24] Seeed Studio, “LoRa-E5 Mini (STM32WLE5JC),” consultado em: 9 abr. 2026. [Online]. Available: https://www.seeedstudio.com/LoRa-E5-mini-STM32WLE5JC-p-4869.html

[25] ——, “LoRa-E5 Dev Kit,” consultado em: 9 abr. 2026. [Online]. Available: https://www.seeedstudio.com/LoRa-E5-Dev-Kit-p-4868.html

[26] ——, “Grove - LoRa-E5 (STM32WLE5JC),” consultado em: 9 abr. 2026. [Online]. Available: https://www.seeedstudio.com/Grove-LoRa-E5-STM32WLE5JC-p-4867.html

[27] RAKwireless, “WisDuo breakout board RAK3272S,” consultado em: 9 abr. 2026. [Online]. Available: https://store.rakwireless.com/products/wisduo-breakout-board-rak3272s

[28] Waveshare, “ESP32-S3-LR1121 development board,” consultado em: 9 abr. 2026. [Online]. Available: https://www.waveshare.com/esp32-s3-lr1121.htm

[29] *STM32L432KC Ultra-low-power Arm Cortex-M4 32-bit MCU Datasheet*, STMicroelectronics, consultado em: 20 jun. 2026. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l432kc.pdf

[30] *LPC1769/68/67/66/65/64/63 Datasheet*, NXP Semiconductors, consultado em: 20 jun. 2026. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/LPC1769_68_67_66_65_64_63.pdf

[31] *SX1276/77/78/79 LoRa Transceiver Datasheet*, Semtech Corporation, consultado em: 20 jun. 2026. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-connect/sx1278

[32] Seeed Studio, “Wio-E5 wireless module,” consultado em: 10 jul. 2026. [Online]. Available: https://www.seeedstudio.com/LoRa-E5-Wireless-Module-p-4745.html

[33] RAKwireless, “RAK3172 wisduo LPWAN module,” consultado em: 10 jul. 2026. [Online]. Available: https://store.rakwireless.com/products/rak3172-lpwan-module

[34] *STM32WL55/54xx Datasheet*, STMicroelectronics, consultado em: 10 jul. 2026. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32wl55jc.pdf

[35] STMicroelectronics, “NUCLEO-L432KC development board,” consultado em: 10 jul. 2026. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-l432kc.html

[36] NXP Semiconductors, “LPCXpresso board for LPC1769 with CMSIS-DAP probe,” consultado em: 10 jul. 2026. [Online]. Available: https://www.nxp.com/design/design-center/development-boards-and-designs/lpcxpresso-board-for-lpc1769-with-cmsis-dap-probe:OM13085

[37] *ESP32-S3 Series Datasheet*, Espressif Systems, consultado em: 20 jun. 2026. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[38] Rain Bird Corporation, “TBOS-BT bluetooth battery-operated irrigation controller,” consultado em: 1 jul. 2026. [Online]. Available: https://www.rainbird.com/professionals/tbos-bt-bluetooth-battery-operated-irrigation-controller

[39] ——, “TBOS-BT/LT bluetooth battery-operated controller technical specification,” consultado em: 1 jul. 2026. [Online]. Available: https://www.rainbird.com/media/8262

[40] ——, “CP/CPF line automatic sprinkler valves,” consultado em: 1 jul. 2026. [Online]. Available: https://www.rainbird.com/products/cp-cpf-line-automatic-sprinkler-valves