Universidade Federal de Ouro Preto
Escola de Minas
CECAU - Colegiado do Curso de
Engenharia de Controle e Automação

Alex Júnior Guimarães

Gateway IoT usando Raspberry Pi

Monografia de Graduação

Ouro Preto, 2024

Alex Júnior Guimarães

Gateway IoT usando Raspberry Pi

Trabalho apresentado ao Colegiado do Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Universidade Federal de Ouro Preto

Orientador: Prof. Alan Kardek Rêgo Segundo, Dr.

Ouro Preto

2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

G963g Guimaraes, Alex Junior.
Gateway IoT usando Raspberry Pi. [manuscrito] / Alex Junior Guimaraes. - 2024.
79 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Alan Kardek Rêgo Segundo.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. Internet das coisas. 2. Minicomputadores - Raspberry PI. 3. Hypertext Transfer Protocol (HTTP). 4. Long Range (LoRa). 5. Message Queuing Telemetry Transport (MQTT). 6. IEEE 802.11 - Wi-Fi. 7. Representational State Transfer (REST). I. Rêgo Segundo, Alan Kardek. II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita -CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Alex Júnior Guimarães

Gateway IoT usando Raspberry Pi

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 11 de outubro de 2024

Membros da banca

Dr. Alan Kardek Rêgo Segundo - Orientador (Universidade Federal de Ouro Preto)
Dra. Adrielle de Carvalho Santana - Convidado (Universidade Federal de Ouro Preto)
Dr. Bruno Nazário Coelho - Convidado (Universidade Federal de Ouro Preto)

Alan Kardek Rêgo Segundo, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 16/10/2024

Documento assinado eletronicamente por **Alan Kardek Rego Segundo**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 16/10/2024, às 15:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0796073** e o código CRC **AADBDD58**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Agradecimentos

Agradeço primeiramente à minha mãe, Maria Aparecida, e ao meu pai, Raimundo Dionísio, que teve a alegria em vida de compartilhar minha entrada na universidade e hoje descansa de todas as suas lutas. Sem o apoio e os esforços diários deles, nada seria possível, e por isso sou eternamente grato.

Agradeço ao meu amigo Iuri da Silva Diniz pelos vários anos de parceria e apoio. Sou grato também por todas as novas amizades feitas durante o curso, em especial aos meus colegas Ary Trindade, Ray Basílio e Davi Pereira.

Agradeço ao meu orientador, Alan Kardek Rêgo Segundo, pelos diversos ensinamentos, auxílios e paciência durante a produção deste trabalho.

Agradeço aos meus antigos colegas de trabalho e hoje grandes amigos, Joyce Oliveira e Thyeres Canuto. Este trabalho tem inspiração direta nos ensinamentos passados por esses dois grandes profissionais da Telecom.

Por fim, agradeço a todo o corpo docente do DECAT (Departamento de Controle e Automação), cada um tendo papel fundamental no meu desenvolvimento. Espero um dia ter a capacidade de honrar os ensinamentos transmitidos.

Resumo

A Internet das Coisas (IoT) vem revolucionando a abordagem tecnológica sobre a visão social e industrial. Em constante evolução, se caracteriza pela forte dependência de conexões sem fio (*wireless*). Entretanto, uma única aplicação IoT pode exigir diferentes técnicas para atender diversos requisitos de conexão, levando ao emprego de tecnologias inicialmente incompatíveis entre si. Por essa razão, este trabalho propõe uma arquitetura de Gateway IoT independente com Raspberry Pi 3B+ capaz de controlar e intercomunicar uma rede heterogênea de dispositivos para armazenamento, controle e suporte em integração de aplicações IoT dos usuários via API REST. Atualmente utilizando HTTP, LoRa e MQTT, o Gateway mostrou-se uma opção viável, oferecendo abstração entre as modalidades e simplificando a construção de redes IoT. Isso foi possível graças à implementação de uma plataforma Web de gerenciamento intuitiva e estruturas básicas predefinidas, que facilitam o registro de nós, estabelecimento de conexões, comunicação entre dispositivos e acesso eficiente às informações, proporcionando uma solução prática para a complexidade das redes IoT heterogêneas.

Palavras-chaves: IoT, Raspberry PI, HTTP, LoRa, MQTT, Wi-Fi, REST.

Abstract

The Internet of Things (IoT) has been revolutionizing the technological approach to both social and industrial visions. In constant evolution, it is characterized by a strong dependence on wireless connections. However, a single IoT application may require different techniques to meet various connection requirements, leading to the use of technologies that were initially incompatible with each other. For this reason, this work proposes an independent IoT Gateway architecture using Raspberry Pi 3B+, capable of controlling and intercommunicating a heterogeneous network of devices for storage, control, and support in the integration of users' IoT applications via REST API. Currently using HTTP, LoRa, and MQTT, the Gateway has proven to be a viable option, offering abstraction between modalities and simplifying the construction of IoT networks. This was made possible through the implementation of an intuitive web management platform and predefined basic structures, which facilitate node registration, connection establishment, communication between devices, and efficient access to information, providing a practical solution for the complexity of heterogeneous IoT networks.

Key-words: IoT, Raspberry PI, HTTP, LoRa, MQTT, Wi-Fi, REST.

Lista de ilustrações

Figura 1 – IoT como Rede das Redes 18
Figura 2 – Blocos Básicos da IoT 19
Figura 3 – Fluxo de envio com o modelo OSI entre dois *hosts* 21
Figura 4 – Processo de Encapsulamento no modelo OSI 21
Figura 5 – Modelagem em camadas do TCP/IP 23
Figura 6 – Aplicação de tecnologias *wireless* na concepção de redes IoT 24
Figura 7 – Esquemático de uma RSSF 24
Figura 8 – Exemplo de sinal Wi-Fi Atenuado 25
Figura 9 – Topologia Wi-Fi IEEE 802.11 26
Figura 10 – Espectro de potência de sinais 27
Figura 11 – Transmissão de símbolos Chirp com SF10 28
Figura 12 – Espectro ISM liberado de autorização no Brasil 28
Figura 13 – Requisição HTTP Cliente/Servidor 30
Figura 14 – Partes de uma mensagem HTTP 31
Figura 15 – Arquitetura de comunicação MQTT 33
Figura 16 – Estrutura de envio de mensagens em redes MQTT 34
Figura 17 – Raspberry PI Modelo 3b+ 36
Figura 18 – Aplicação recebendo um objeto JavaScript (JSON) pela requisição HTTP GET em uma API REST 37
Figura 19 – Arquitetura de um Gateway IoT residencial 38
Figura 20 – Informações de um dispositivo no Gateway via REST 39
Figura 21 – Arquitetura do Gateway IoT 40
Figura 22 – Montagem do Raspberry Pi 3B+ como Gateway 41
Figura 23 – Servidor Web - Página inicial 42
Figura 24 – Sistema de Login 43
Figura 25 – Troca de credenciais do Sistema 44
Figura 26 – Cadastro de nós no Gateway 45
Figura 27 – Lista de nós MQTT cadastrados 46
Figura 28 – Página de gerência da Interface HTTP 46
Figura 29 – Página de gerência da Interface LoRa 47
Figura 30 – Página de gerência da Interface MQTT 47
Figura 31 – Página de configuração das rotas de repetição 48
Figura 32 – *Log* do sistema no tratamento de recepção de pacotes 49
Figura 33 – Requisição de mensagens de um nó no Gateway com o Postman 51
Figura 34 – Adição do código NAP na requisição de mensagens no Postman 51
Figura 35 – Broker Eclipse Mosquitto no Raspberry Pi 52

Figura 36 – Memória RAM reservada para cache no MongoDB 53
Figura 37 – Mensagens de nós armazenadas no MongoDB 53
Figura 38 – Exemplo típico de endereçamento NID 55
Figura 39 – Comunicação HTTP entre Nós e Gateway 56
Figura 40 – Bloco esquemático da família SX127x 58
Figura 41 – Cabeçalho de pacote LoRa 59
Figura 42 – Comunicação LoRa entre Nós e Gateway 60
Figura 43 – Circuito esquemático de conexão dos módulos LoRa 61
Figura 44 – Montagem dos rádios LoRa no Raspberry Pi 61
Figura 45 – Comunicação MQTT entre Nós e Gateway 62
Figura 46 – Montagem dos nós de rede 64
Figura 47 – Circuito esquemático de conexão do LoRa XL1276-P01 no Wemos Mini D1 64
Figura 48 – Exemplo de um *dashboard* Grafana 65
Figura 49 – *Dashboard* Grafana “Rasp4IoT Nodes” 66
Figura 50 – Requisições GET na API REST via Grafana 66
Figura 51 – Cadastro dos nós Wemos Mini D1 67
Figura 52 – Configuração de endereçamento 68
Figura 53 – Rotas de comunicação entre os nós Wemos Mini D1 69
Figura 54 – *Log* de comunicação do Wemos Mini D1 - HTTP 70
Figura 55 – *Log* de comunicação do Wemos Mini D1 - LoRa 70
Figura 56 – *Log* de comunicação do Wemos Mini D1 - MQTT 70
Figura 57 – Gráfico de mensagens dos nós na rede 71
Figura 58 – Desempenho do Raspberry Pi como Gateway 71

Lista de tabelas

Tabela 1 – Características do 2AD66-LORAV2 (SX1276) 57
Tabela 2 – Parâmetros de identificação dos nós na rede 68

Lista de abreviaturas e siglas

6LoWPAN	IPv6 over Low-Power Wireless Area Network
API	Application Programming Interface
BLE	Bluetooth Low Energy
CAD	Carrier Activity Detection
CCA	Clear Channel Assessment
CR	Coding Rate
CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
DSSS	Direct Sequence Spread Spectrum
DPWS	Device Profile for Web Services
FEC	Forward Error Correction
FI	Frequência Intermediária
FHSS	Frequency Hopping Spread Spectrum
FPGA	Arranjo de Porta Programável em Campo (Field Programmable Gate Array)
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
HAT	Hardware Attached on Top
HTTP	HyperText Transfer Protocolo
IoT	Internet das Coisas (Internet of Things)
ISM	Industrial, Scientific, and Medical
ISO	International Organization for Standardization
LNA	Low Noise Amplifier
LoRa	Long Range

LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
MAC	Endereço de Controle de Acesso à Mídia (Media Acess Control)
MQTT	Message Queue Telemetry Transport
NAP	Network API
NID	Network Identifier
OSI	Open System Interconnection (Interconexão de Sistemas Abertos)
PLL	Phase-Locked Loop
PoE	Power over Ethernet
PUBACK	Publish Acknowledged
PUBCOMP	Publish Complete
PUBREC	Publish Received
PUBREL	Publish Release
PINGREQ	Ping Request
PINGRESP	Ping Response
QoS	Quality of Service
RF	Rádiofrequência
RSSI	Received Signal Strength Indication
SF	Spreading Factor
SMTP	Simple Mail Transfer Protocol
SS	Spread Spectrum
TCP/IP	Protocolo de Controle de Transmissão / Protocolo de Internet
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier

REST	Representational State Transfer
RSSF	Redes de Sensores Sem Fio
WLAN	Wireless Local Area Network
WWW	World Wide Web

Sumário

1 INTRODUÇÃO 15
1.1 Justificativas e Relevância 16
1.2 Objetivos 16
1.3 Metodologia 17

2 FUNDAMENTAÇÃO TEÓRICA 18
2.1 Internet das Coisas 18
2.2 Modelo OSI 20
2.3 Protocolo TCP/IP 22
2.4 Redes de Sensores Sem Fio - RSSF 23
2.4.1 Wi-Fi WLAN - IEEE 802.11 25
2.4.2 LoRa 26
2.5 HTTP 29
2.5.1 Solicitações HTTP com URL 30
2.5.2 Estrutura das mensagens HTTP 31
2.5.3 Métodos HTTP 31
2.5.4 Códigos de Status 32
2.6 MQTT 32
2.6.1 Conceitos e Recursos Básicos 33
2.7 Raspberry Pi 36
2.8 RESTful API em IoT 37

3 DESENVOLVIMENTO 38
3.1 Arquitetura do Sistema 38
3.2 Node.js 41
3.3 Servidor Web 42
3.3.1 Sistema de Autenticação 43
3.3.2 Cadastro de Dispositivos 44
3.3.3 Gerência das Interfaces 45
3.3.4 Roteamento de mensagens 48
3.3.5 Sistema de consulta de mensagens 49
3.4 Servidor MQTT 50
3.5 Servidor Banco de Dados 51
3.6 Interfaces de Rede 54
3.6.1 Sistema de Endereçamento - NID 54
3.6.2 Interface HTTP 54

3.6.3 Interface LoRa 56
3.6.4 Interface MQTT 61
3.7 Nós de Rede 63
3.7.1 Servidor Grafana 64

4 RESULTADOS 67
4.1 Ativação dos nós 67
4.2 Configuração dos Wemos Mini D1 68
4.3 Testes de intercomunicação 69
4.4 Consulta de mensagens via API 69
4.5 Desempenho do Raspberry Pi 71

Conclusão 73
4.6 Sugestões de trabalhos futuros 74

Referências 75

1 Introdução

Os sistemas IoT, conhecidos como Internet das Coisas (*Internet of Things*), é uma aplicação já estabelecida e utilizada nas redes de computadores. É o que defendem Santos et al. (2016), enfatizando que o desenvolvimento veio graças à evolução tecnológica de áreas relacionadas a sistemas embarcados, eletrônica, sensores e telecomunicação, chamando a atenção tanto da indústria quanto da comunidade acadêmica. Islam, Uddin e Kwak (2016) demonstram a versatilidade que as aplicações IoT possuem, podendo ser usadas em sistemas de transporte inteligentes, comunicações, eletrodomésticos, medicina, serviço de vigilância e controle de aplicações industriais.

A Internet atual, aborda Comer (2016), é baseada em conjuntos de várias redes interconectadas trocando dados entre si, possibilitando a adição de novas tecnologias sem que as antigas sejam descartadas. Esse estágio foi alcançado pela elaboração da pilha de protocolos TCP/IP (*Transmission Control Protocol / Internet Protocol*), que segundo Comer (2016), foi o resultado da falta de um meio em comum de envio de informação que não focasse em tecnologias específicas. Desta maneira, o TCP/IP constituiu alternativas para estabelecer um padrão de operação geral independente no qual características virtuais como tamanho de pacotes são mais relevantes do que os aspectos de *hardware* que estão sujeitos aos fabricantes.

Um fator importante para o uso das redes IoT é a comunicação sem fio (*wireless*). É o que relatam Van Kranenburg e Bassi (2012), apontando que esta modalidade de conexão permitiu que gateways IoT e sensores pudessem se comunicar distanciados entre si, tendo em comum os diversos padrões sem fio existentes e que devem ser escolhidos conforme vários requisitos de projeto. Entre elas, podem ser citados o alcance, largura de banda e consumo de energia, levando em conta a possibilidade de instalações em áreas de difícil acesso e manutenção. Essas condições requerem o uso de redes confiáveis e que podem depender de fontes renováveis alternativas para garantir o sucesso da operação, a exemplo dos painéis solares.

Existem várias alternativas ao estabelecer enlaces IoT para conectar nós. Elkhodr, Shahrestani e Cheung (2016) apresentam uma discussão de algumas tecnologias comuns na concepção de projetos IoT por prover métodos de integração com baixo custo e gasto de energia, entre eles o ZigBee, 6LoWPAN (*IPv6 over Low-Power Wireless Area Network*), BLE (*Bluetooth Low Energy*), LoRa (*Long Range*), LoRaWan e Wi-Fi. Apesar disso, ressaltam que essas características exigem observações quanto à segurança e privacidade das informações que os sistemas trafegam, o que implica no desafio de implementar um sistema de gerenciamento e controle que propicia um ambiente heterogêneo de transmissão

de pacotes, ou seja, que suporte várias tecnologias, mas ao mesmo tempo seja estável e protegido.

1.1 Justificativas e Relevância

A constante evolução dos sistemas de informação abriu caminho para sistemas IoT se apresentarem como solução de crescimento do dinamismo da coleta de dados. No entanto, citam Van Kranenburg e Bassi (2012), a sua implementação trouxe novos desafios que chegam até mesmo em discussões éticas como controle social e consentimento em captura de dados pessoais. Na parte tecnológica, foco deste trabalho, as dificuldades estão diretamente associadas à diversidade de nós inteligentes, que a exemplo de Chen, Jia e Li (2011) discorrem, estão interconectados em redes que ao mesmo tempo podem ter dispositivos de curto e longo alcance utilizando, diferente da internet tradicional, diversas técnicas de comunicação para usos específicos. Contudo, defendem que a proposta de um Gateway IoT padronizado pode acelerar consideravelmente a elaboração de novas soluções.

Com a Internet das Coisas se estabelecendo e evoluindo a forma como os dispositivos eletrônicos são desenvolvidos, reflexões e preparações precisam ser realizadas para evitar déficit tecnológico. Pereira e Oliveira Simonetto (2018) apontam que esse avanço atinge os processos de produção, abrindo caminho para a Indústria 4.0. Portanto, é preciso que tanto os setores produtivos quanto os institutos de ensino se preparem a esta revolução que tende a mudar a maneira com que as pessoas trabalham e se socializam. Deve-se observar, entretanto, que desenvolver aplicações é custoso e muitas vezes requer grandes equipes. Em seus estudos, Flores, Ribeiro e Echeverria (2017) abordam sobre as dificuldades do uso da Tecnologia da Informação e Comunicação (TIC) no ensino superior, com destaque a falta de cursos de formação para uso pedagógico, poucos computadores e tempo escasso dos professores. Pensando nisso, é importante propor estratégias que facilitem a forma que as tecnologias são empregadas, facilitando a preparação dos indivíduos tanto nos centros de estudos quanto em suas futuras experiências profissionais.

1.2 Objetivos

Este trabalho tem como objetivo geral a implementação de um Gateway (concentrador) capaz de gerenciar uma rede de comunicação entre microcontroladores utilizando rede sem fio (*wireless*) para tráfego e armazenamento de dados de aplicações IoT como sensores e demais sistemas embarcados. Paralelamente para apoio e direcionamento, há os seguintes objetivos específicos:

- Proporcionar um ambiente prático e de fácil acesso para gerenciar uma rede de

dispositivos remotos;

- Desenvolvimento de um protótipo capaz de ser heterogêneo no quesito de suporte a intercomunicação de diferentes tecnologias de comunicação de rede ao mesmo tempo;
- Disponibilização dos dados dos dispositivos na Internet via interface Web para integração em aplicações externas;
- Fazer com que o Gateway seja capaz de ser independente de outros serviços e aplicações para estabelecer a rede.

1.3 Metodologia

A construção deste estudo foi baseada em pesquisas científicas que possibilitassem propor uma alternativa com potencial de facilitar o desenvolvimento e avanço das redes IoT. Com esse intuito, é apresentado o uso do sistema embarcado Raspberry Pi 3 B+ como um Gateway dotado de conexão de módulos sem fio LoRa e Wi-Fi (incluso) capaz de receber dados de dispositivos diversos como sensores e atuadores.

O trabalho visa, a partir de estudos de aplicação de tecnologias *wireless*, encontrar uma maneira apropriada para além de receber dados, armazená-los e apresentá-los para serem utilizados na Internet. O foco principal é criar uma solução integrada que facilite a comunicação entre uma ampla gama de equipamentos, desde sensores industriais até dispositivos domésticos inteligentes. Com esse propósito, a pesquisa se dedicou à estruturação dos elementos necessários para cumprir os objetivos, como escolhas de *hardware* dos dispositivos, maneiras de padronizar mensagens, o banco de dados para armazenamento das informações e o uso de um servidor Web para apresentação e gerência da aplicação pelos usuários. Todos esses parâmetros buscam atender limitações físicas do Raspberry Pi e garantir um registro confiável das informações recebidas. Por fim, os resultados obtidos foram analisados, permitindo concluir se o projeto atende aos critérios desejados.

O arranjo desta monografia é realizado focando na divisão entre capítulos. No 1 tem-se a introdução com uma breve discussão do tema e apresentação da justificativa, objetivos, motivação na escolha do assunto e a metodologia aplicada. Em sequência é apresentada o capítulo 2, abordando resumidamente a teoria por trás dos principais tópicos relevantes ao campo que este trabalho se inclui. Por fim, nos capítulos 3 e 4, são relatados o desenvolvimento, as análises e disponibilizações dos resultados alcançados para as conclusões sobre o trabalho.

2 Fundamentação Teórica

2.1 Internet das Coisas

A Internet das Coisas, ou Internet dos Objetos, como cita Evans (2011), é uma evolução da atual Internet, sendo hoje formada por uma rede de redes, tal como representado na Figura 1. Isso é ressaltado pela exemplificação em automóveis, com a união de redes distintas para controlar recursos do motor, alarme, comunicação, entre outros subsistemas. Essa conexão de módulos distintos é o que dá origem ao IoT.

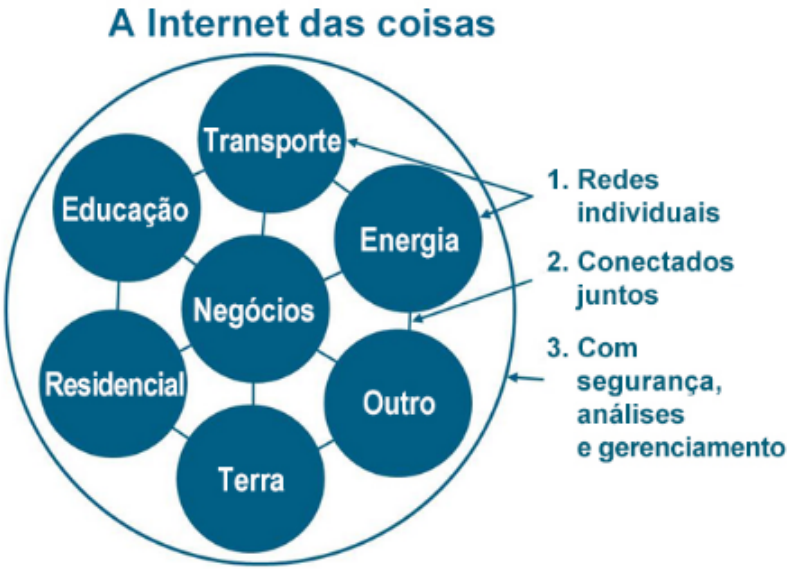

Figura 1 – IoT como Rede das Redes. Fonte: Adaptado de Evans (2011, p. 5).

Santos et al. (2016) defendem que essas redes podem ser construídas por blocos, como mostra a Figura 2, que unidos, ajudam a tornar possível a integração das mais diversas tecnologias. São eles:

- **Identificação**: responsável por detectar dispositivos, necessário para conexão em rede na Internet, como no caso de endereçamento IP;
- **Sensores/Atuadores**: sensores como ferramenta de captura de dados para futuro envio e armazenamento, e os atuadores como interfaces para modificar o comportamento de sistemas conforme comandos ou dados recebidos;
- **Comunicação**: os meios para estabelecer um barramento para tráfego de informação. Podem ser citados o Wi-Fi e Bluetooth, sendo importante considerar o consumo elétrico.

- **Computação**: as unidades que ficam responsáveis pelo processamento de dados dentro dos objetos, a exemplo os microcontroladores e FPGA.
- **Serviços**: podem ser utilizados para vários tipos de atividades, destacando-se:
  - **Identificação**: captura característica físicas de um ambiente (Entidades Físicas) e a transforma em dados (Entidades Virtuais);
  - **Agregação de Dados**: captura e sumarização de dados;
  - **Colaboração e Inteligência**: processamento sobre dados de agregação para tomada de decisão;
  - **Ubiquidade**: proporcionar disponibilidade constante e de qualquer lugar aos serviços de colaboração e inteligência.
- **Semântica**: táticas para extração de material relevante em conjuntos de dados no intuito de fornecer um serviço.

Figura 2 – Blocos Básicos da IoT. Fonte: Santos et al. (2016, p. 6).

Com avanços da IoT, vai se tornar possível, defendem Santos et al. (2016), criar dispositivos com inteligência que podem ser monitorados, controlados, trocar informação entre si e até mesmo interagir com pessoas. Assim sendo, propicia a existência de Cidades Inteligentes (*Smart Cities*), Casas Inteligentes (*Smart Home*) e avanços na medicina de Assistência Médica (*Health Care*). Por outro lado, ainda há pontos de atenção, como as dificuldades na interpretação e consistência de dados. Em consequência, é indispensável considerar erros oriundos de calibração de sensores, dados fora de ordem, corrompidos ou ruidosos (*outliers*), correta detecção de transmissores, e diversos outros.

2.2 Modelo OSI

Considerando a diversidade de tecnologias de rede existentes, a integração para estabelecer comunicação poderia ser um desafio. Alani (2014) retrata que em meados dos anos 70, a falta de padronização trouxe para a ISO (*International Organization for Standardization*), Organização Internacional de Padronização, pedidos de discussão de propostas de uma arquitetura padronizada e unificada para sistemas de processamento distribuído. Atendendo a essas solicitações, surgiu em 1979 o modelo de referência de Interconexão de Sistemas Abertos, conhecido como OSI (*Open System Interconnection*).

O uso do OSI para explicar protocolos de comunicação é rotineiro, explica Hunt (2002), graças a difusão dos seus conceitos na comunidade de comunicação de dados. Li et al. (2011) ressaltam que o OSI tem foco em solucionar problemas de compatibilidade de redes heterogêneas, destacando a capacidade que ele possui em separar serviços, interfaces e protocolos.

Hunt (2002) esclarece que o modelo OSI é constituído de sete níveis de camadas para definir as atribuições dos protocolos de comunicação, e não o protocolo em si. Uma infinidade de protocolos são possíveis para uma mesma camada, entretanto, eles só se comunicam com seu correspondente no lado receptor, precisando apenas conhecer o método de envio para as camada seguintes.

A Figura 3 mostra como são dispostas as camadas na arquitetura OSI em uma comunicação hipotética entre dois dispositivos (*hosts*). Alani (2014) descreve resumidamente as setes camadas do modelo OSI como apresentado a seguir:

- **Física (*Physical*)**: especificação de características físicas (sinal, conector, meio físico, entre outros) para controle de transmissão de bits nas interfaces;
- **Enlace (*Data Link*)**: armazenamento de informações de controle em quadros (*frames*) para funções associadas ao estabelecimento de comunicação confiável, incluindo detecção e correção de erros, identificação, estabelecimento de *link*, reencaminhamento e controle de sequência de *frames*;
- **Rede (*Network*)**: dados em pacotes responsáveis pelas operações de roteamento, como escolha da melhor rota, fragmentação de informações, endereçamento lógico, sequenciamento, detecção e recuperação de erros;
- **Transporte (*Transport*)**: contém recursos para garantir o transporte de ponta a ponta em conexões orientadas (conexão deve anteceder o início da transmissão, como em chamadas telefônicas) e não orientadas (envio sem confirmação de entrega, como no envio de correspondências), servindo de apoio para a camada de Sessão;

- **Sessão (*Session*)**: gerenciamento de recursos para organização de múltiplas sessões de comunicação;
- **Apresentação (*Presentation*)**: apresentação dos dados para a camada de aplicação, incluindo serviços de criptografia, compressão e tradução;
- **Aplicação (*Application*)**: definição dos serviços usados nas aplicações do usuário final na rede.

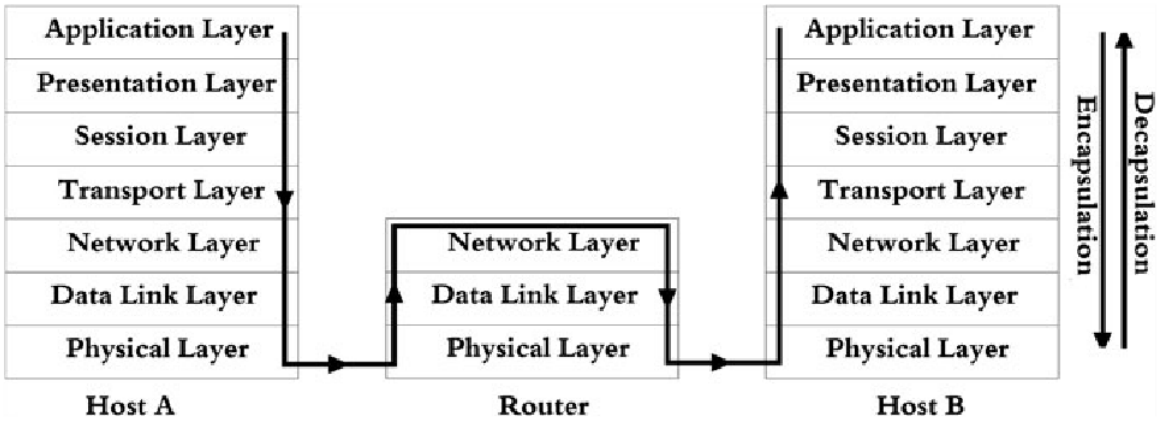

Figura 3 – Fluxo de envio com o modelo OSI entre dois *hosts*. Fonte: Alani (2014, p. 15).

Pode-se observar na Figura 3 os processos de encapsulamento e desencapsulamento entre o *host* A e B com a participação de um roteador. Alani (2014) relata que na transmissão, o *host* A monta o pacote adicionando as informações de controle junto com a informação desde a camada de aplicação até a física, resultando no processo de encapsulamento mostrado na Figura 4. Por outro lado, o *host* B realiza o processo inverso, com cada camada recebendo o dado referente a ela até que o dado seja entregue para a aplicação do receptor. O roteador agirá apenas até a camada de Rede, pois ela permitirá identificar o endereço do destinatário, que é o seu objetivo, já que sua função se resume ao roteamento.

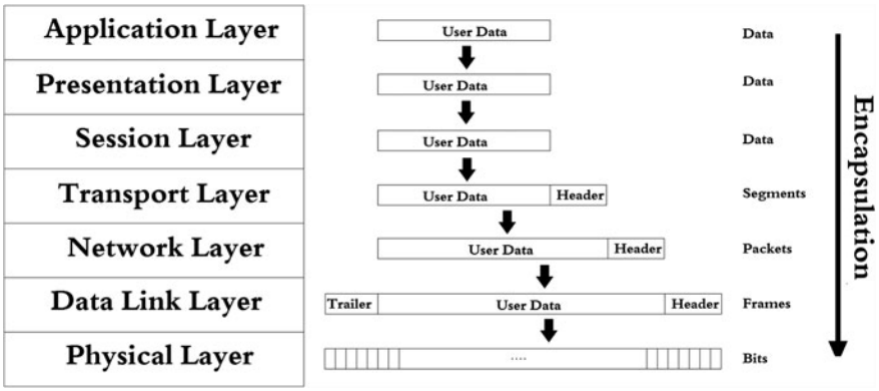

Figura 4 – Processo de Encapsulamento no modelo OSI. Fonte: Alani (2014, p. 15).

2.3 Protocolo TCP/IP

O TCP/IP é a união do Protocolo de Controle de Transmissão (*TCP - Transmission Control Protocol*) com o Protocolo de Internet (*IP - Internet Protocol*). É o que elucida Hunt (2002), acrescentando a característica de proporcionar comunicação transparente com diversos tipos de *hardware* e sistemas operacionais e ainda ser uma suíte de protocolos aberta, gratuita e padronizada. Semelhante ao modelo OSI, o TCP/IP também é arquitetado em camadas com níveis de protocolos como apresentado na Figura 5.

A diferença sobre o modelo OSI fica pelas funções e número de camadas, restritas a quatro. Parziale et al. (2006) explica que elas possuem funções específicas e independentes, utilizando interfaces para se comunicar com as superiores e inferiores da pilha de protocolos. Elas podem ser descritas como segue:

- **Aplicação (*Applications*)**: programa do usuário que utiliza comunicação TCP/IP para acesso a outro *host* usando portas ou *sockets* como interface com a camada de Transporte. Exemplos incluem HTTP, SSH, Telnet, entre outros;
- **Transporte (*Transport*)**: proporciona a transmissão de informações das aplicações para um *host* na rede. São comuns o uso do TCP ou UDP (Protocolo de Datagrama do Usuário - *User Datagram Protocol*). A diferença principal entre eles é como tratam o envio da informação. O TCP é orientado a conexão, com recursos de garantia de entrega segura, enquanto o UDP é não orientado, ou seja, não necessita conexão. O UDP não possui confirmação de recebimento ou integridade, mas é útil em aplicações que demandam velocidade com margem de perda aceitável de dados;
- **Inter-rede (*Internetwork*)**: envolve os protocolos para gerência da rede lógica, destacando-se o protocolo IP por fornecer o endereçamento de máquinas e roteamento de pacotes. Outros exemplos são o ICMP (Protocolo de Mensagens de Controle da Internet - *Internet Control Message Protocol*), fornecendo recursos para análise (ping, traçar rota, entre outros) e o DHCP (Protocolo de Configuração Dinâmica de *Host* - *Dynamic Host Configuration*) para autoconfiguração de endereços em novos dispositivos integrantes em uma rede;
- **Interface de Rede e *Hardware* (*Network Interface and hardware*)**: compreende os protocolos que fazem a interface com os mais diversos *hardwares* de rede possíveis, podendo incluir recursos de entrega confiável (detecção de erros, sequenciamento, confirmação de chegada) ou não. Podem ser mencionados o Ethernet, fibra óptica (FDDI) e linhas telefônicas (DSL).

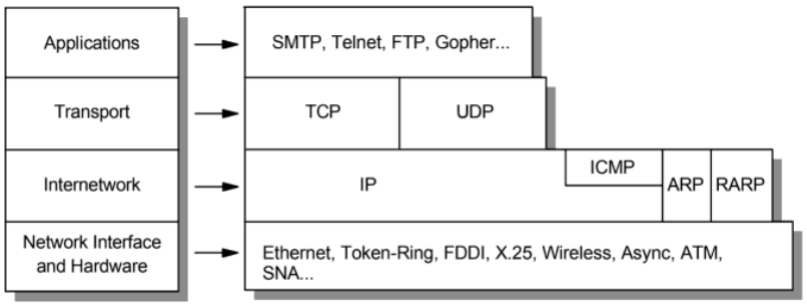

Figura 5 – Modelagem em camadas do TCP/IP. Fonte: Parziale et al. (2006, p. 9).

Dois conceitos relevantes na camada de Transporte são os termos de portas e soquetes (*sockets*). Conforme Parziale et al. (2006) retratam, as portas são um número de 16 bits que os programas utilizam para identificar a aplicação TCP. Alguns serviços possuem portas padronizadas, como os servidores HTTP (80) e FTP (21), mesmo que não seja obrigatório ou imutável. Como resultado, mesmo o TCP/IP sendo um protocolo orientado a conexão de ponto a ponto (não exige chefe/subordinado), as aplicações são cliente/servidor. Isso implica que o cliente utiliza uma porta em sua aplicação e faz uma requisição para a correspondente no servidor, aguardando uma resposta.

Quanto aos *sockets*, Parziale et al. (2006) complementam que eles são uma interface genérica de programação e comunicação. O *socket* fornece uma conexão *full-duplex* entre aplicações (bi-direcional simultânea) utilizando uma porta TCP. Esse recurso permite que as aplicações aproveitem as vantagens de uma conexão orientada (confiabilidade, ordem de envio garantida, detecção de erros) enquanto interagem entre si, isso desde que o cliente inicialmente solicite o estabelecimento do *socket* com o servidor.

2.4 Redes de Sensores Sem Fio - RSSF

O enlace de comunicação entre equipamentos dentro de uma rede pode ser feito tanto via cabos quanto sem fio. Rufino (2019) evidencia que apesar das redes cabeadas terem a vantagem de possuírem recursos físicos como o suporte a proteção ao meio externo, as redes sem fio conseguem atingir locais de difícil acesso que muitas vezes impossibilitam o emprego de condutores. No entanto, ressalta que elas necessitam de maiores precauções com relação à segurança da informação. A Figura 6 ilustra um possível cenário real com redes de tecnologias heterogêneas conectando dispositivos diversos.

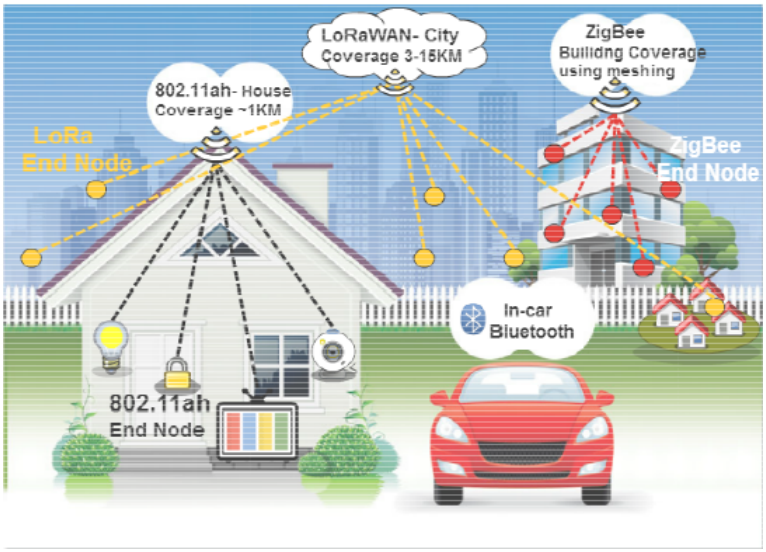

Figura 6 – Aplicação de tecnologias *wireless* na concepção de redes IoT. Fonte: Elkhodr, Shahrestani e Cheung (2016, p. 77).

No caso da aplicação com nós sensores, pode ser adotada a Rede de Sensores Sem Fio (RSSF), exemplificada na Figura 7. Gomes et al. (2014) explicam que elas possuem a vantagem de serem mais fáceis de implementar, manter e ainda têm baixo custo. Porém, é preciso atentar para os problemas de transmissão relacionadas a interferência eletromagnética, ruído de máquinas elétricas ou nós vizinhos, e também para os requisitos de baixo consumo, velocidade de processamento e baixas taxas de transmissão. Portanto, salientam Gomes et al. (2014), o emprego das RSSF deve considerar técnicas para uso mais inteligente do espectro eletromagnético, principalmente pela maior demanda de sistemas que empregam comunicação *wireless*. A Figura 8 mostra um exemplo de sinal de uma rede Wi-Fi IEEE 802.11g com sinal atenuado por um forno micro-ondas.

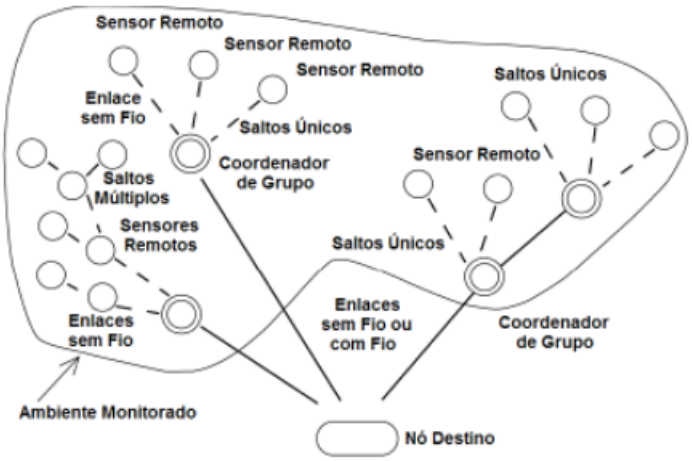

Figura 7 – Esquemático de uma Rede de Sensores Sem Fio (RSSF). Fonte: Sousa e Lopes (2011, p. 41).

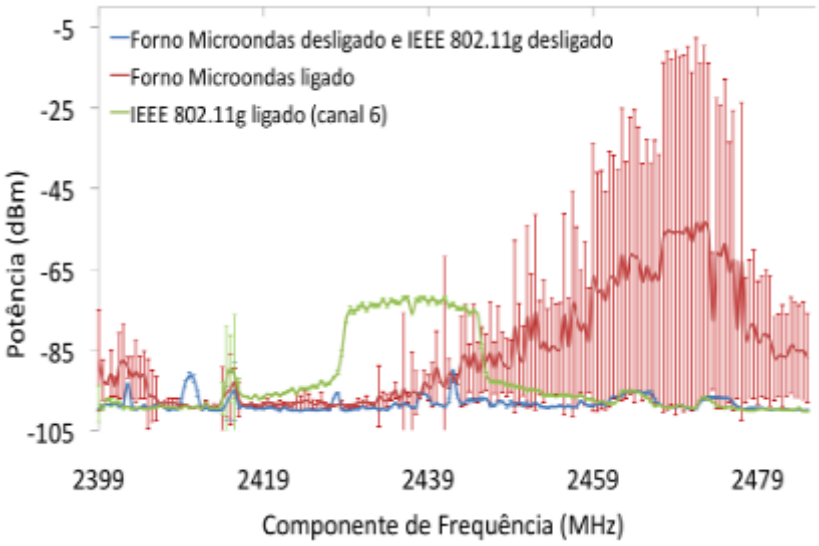

Figura 8 – Exemplo de sinal Wi-Fi atenuado por um forno micro-ondas. Fonte: Gomes et al. (2014, p. 2).

2.4.1 Wi-Fi WLAN - IEEE 802.11

O padrão IEEE 802.11 *Wireless Local Area Network* (WLAN), para Banerji e Chowdhury (2013) é um dos mais utilizados atualmente. Os autores apresentam ainda que elas são basicamente compostas de um Ponto de Acesso (AP, *Access Point*) que age como um gateway ou centralizador para vários outros nós que se conectam a ele, as chamadas Estações (STA, *Station*).

Banerji e Chowdhury (2013) dizem que as redes nos primeiros padrões tinham taxa de *throughput* — quantidade máxima de dados que passa por uma interface — de 11 Mbps, mas que com a evolução do padrão passou para cerca de 54 Mbps e com uma área de alcance entre 50 e 100 metros.

Até alcançar estas capacidades, Banerji e Chowdhury (2013) relatam que diversos padrões foram propostos, saindo do projeto inicial IEEE 802 que considerava apenas o Endereço de Controle de Acesso à Mídia, o MAC, até melhorias que consideravam problemas de transmissão derivados da atenuação de sinal, colisão de dados e integração em barramentos com vários nós que exigia um MAC dedicado para cada interface sem fio. O autor cita que desafios como estes fizeram o IEEE 802 ser ajustado, alcançando a primeira versão aprovada, o IEEE 802.11 em 1991. Em seguida, o padrão foi melhorado em relação das distâncias, taxas de transmissão, modulação, faixa de frequência e largura de banda.

A versão 802.11 está representada na Figura 9, mostrando dois pontos de acesso (AP 1 e AP 2) conectando diversas estações sem fio, como celulares e notebooks, em uma rede local cabeada (*Wired LAN*).

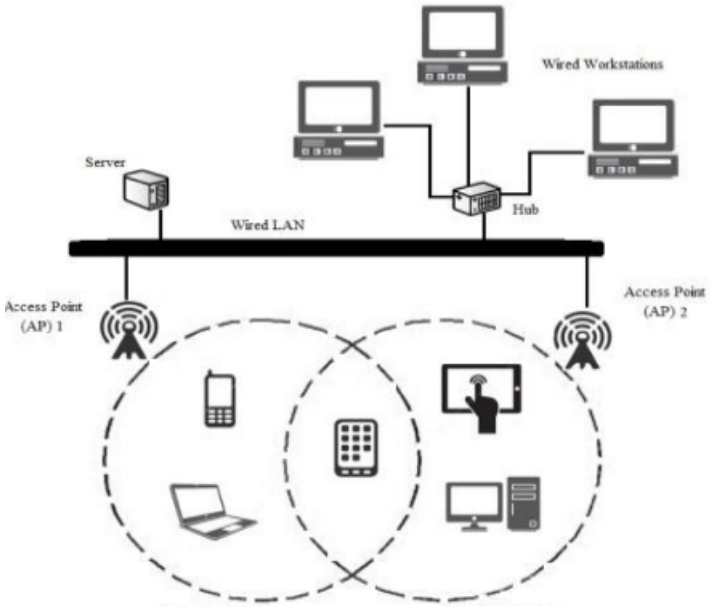

Figura 9 – Topologia Wi-Fi IEEE 802.11. Fonte: Adaptado de Banerji e Chowdhury (2013, p. 2).

2.4.2 LoRa

Outra tecnologia que se enquadra no uso de RSSF (Seção 2.4) é o LoRa, uma técnica baseada em Espalhamento de Espectro (*Spread Spectrum*) proprietária da Semtech Corporation (2019) e baseada no *Chirp Spread Spectrum* (CSS), com o nome derivado de Long Range, tendo como objetivo possibilitar a construção de redes de longo alcance com baixo consumo de energia (LPWAN - *Low Power Wide Area Network*). Essas características de LPWAN, como explicam Ortiz et al. (2019), fazem do LoRa um candidato no uso de redes IoT, atendendo aos requisitos básicos para conexão entre dispositivos inteligentes que necessitam ter gasto mínimo de energia.

McCune (2010) defende que *Spread Spectrum*, ou SS, não é um tipo de modulação, mas sim uma técnica de comunicação que pode ser usada em vários outros tipos de modulações como o *Frequency Hopping Spread Spectrum* (FHSS) e *Direct Sequence Spread Spectrum* (DSSS). McCune (2010) esclarece que o SS é uma forma de dividir um sinal maior em pequenas partes dentro de uma largura de banda, de modo que, a partir de um certo tipo de codificação, o receptor poderá reconstruí-lo. Acrescenta que a sequência de espalhamento pode ser obtida por códigos "*chips*", com deslocamento no tempo pelo "*clock chip*" (relógio chip). Schilling et al. (1991) reforçam que esta característica permite a redução de produção de interferência, uma vez que o princípio de operação consiste em reduzir a potência de um sinal original e espalhá-lo no espectro por um fator de potência, por exemplo, de 100 vezes menor, diminuindo a interferência direta produzida, como exibe a Figura 10.

Segundo Pickholtz, Schilling e Milstein (1982), o SS é amplamente utilizado em

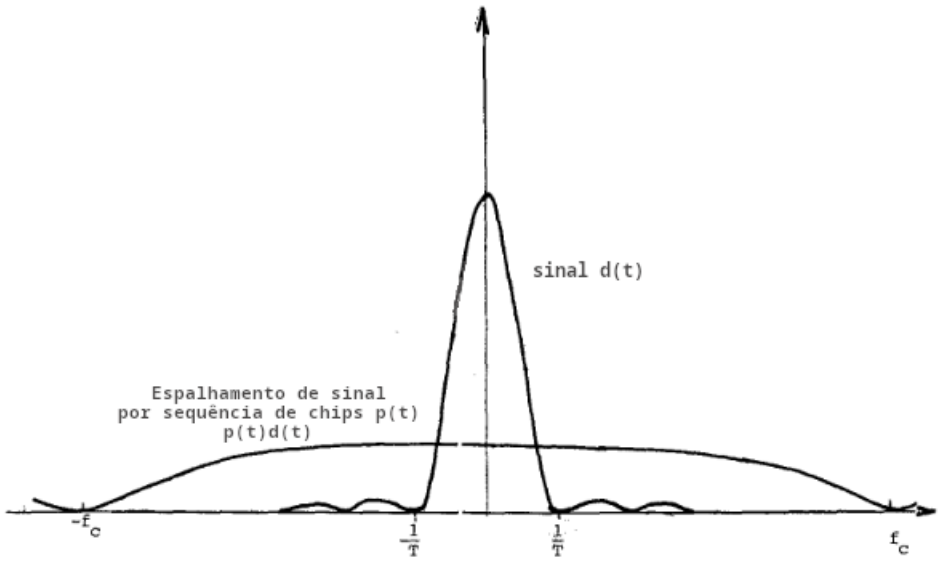

Figura 10 – Exemplo de aplicação de Espalhamento de Espectro. Fonte: Adaptado de Pickholtz, Schilling e Milstein (1982, p. 859).

aplicações militares como ferramenta de proteção contra ataques de bloqueios intencionais devido à sua resistência a interferências e à redução de colisões de sinais. Isso permitiu que mais usuários compartilhassem o espectro de frequências, ainda que a técnica consuma uma fatia maior dele.

No LoRa, explana Montanny (2022), diferente de outras técnicas de *Spread Spectrum* que codificam as informações, o CSS usa símbolos *chirp*. Devalal e Karthikeyan (2018) relatam que o *chirp* é um sinal cuja frequência pode ser incrementada ou decrementada ao longo do tempo, transmitindo um sinal que ocupa uma largura de banda de espalhamento maior do que a largura de banda original necessária para representar a mesma informação. Na Figura 11 pode-se ver um exemplo de transmissão de dados LoRa envolvendo o uso de símbolos *chirp*.

Devalal e Karthikeyan (2018) em seu estudo apresentam que um Gateway LoRa pode transmitir dados em uma área com cerca de 100 km^2, caracterizando uma vantagem em relação aos concorrentes em aplicações que necessitam de baixas taxas de transferência em curtos intervalos de tempo. Bor, Vidler e Roedig (2016) destacam que os rádios LoRa são capazes de operar em diferentes frequências (433 MHz, 868 MHz e 915 MHz), e assim como o padrão IEEE 802.11, possuem a camada física de MAC, no entanto elas só são utilizadas para topologias complexas como o LoRaWAN (*Long Range Wide Area Network*).

Conforme apresentam Bor, Vidler e Roedig (2016) o LoRa é dotado de quatro parâmetros de configuração:

- **Frequência Portadora**: Frequência central (CF, *Carrier Frequency*) da largura de

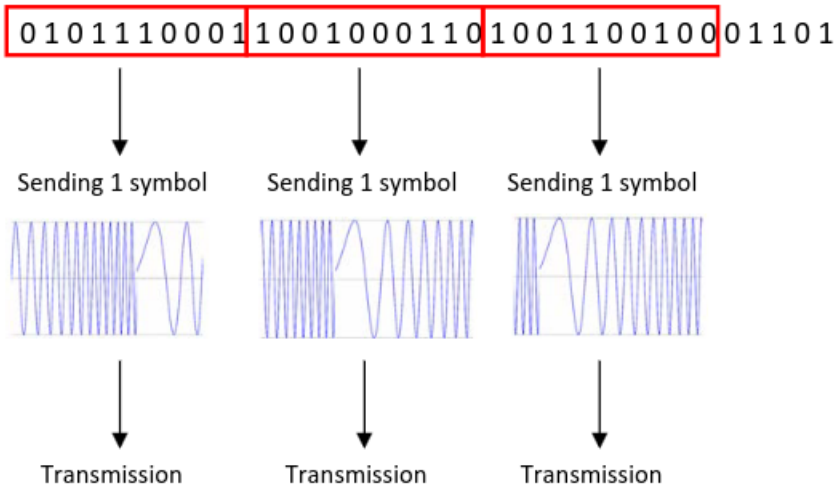

Figura 11 – Transmissão de símbolos Chirp com SF10. Fonte: Montanny (2022, p. 23).

banda escolhida, dependente do módulo LoRa utilizado e das frequências do espectro ISM (*Industries, Scientific and Medic*). É importante observar que segundo a Anatel (2024), apenas a faixa entre 902 MHz e 928 MHz com 915 MHz de faixa central são livres de autorização de uso para uso de aplicações ISM em solo brasileiro, como mostra a Figura 12.

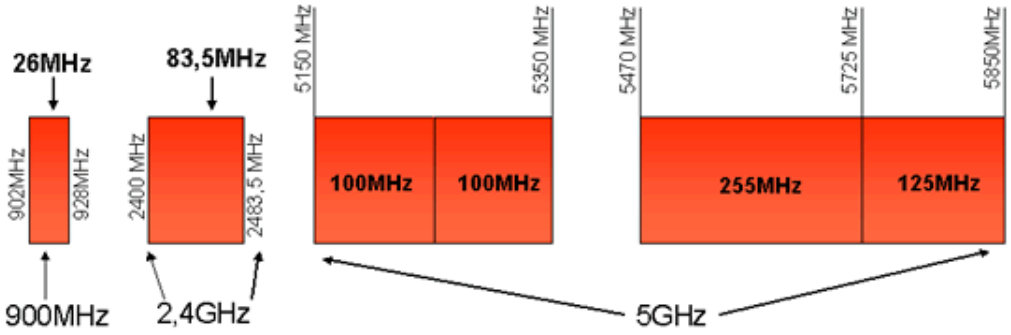

Figura 12 – Espectro ISM livre de autorização no Brasil. Fonte: Teleco (2024).

- ***Spreading Factor***: relação direta com a modulação CSS, é a razão entre as taxas de símbolos transmitidos e a de *chips*, definidas pela relação 2^{SF}. Desta forma, o SF 12 contém 4096 chips por símbolo. Quanto menor o SF, mais rápido se dá a entrega de dados, por outro lado, o SF maior tem a vantagem de elevar a relação sinal-ruído (SNR), a sensibilidade e o alcance, mas com a desvantagem de alongar o tempo de envio, o que por consequência, aumenta o consumo de energia. Os rádios suportam o SF entre SF6 e SF12, sendo o primeiro um caso especial. Um destaque é que transmissões com diferentes SF são ortogonais entre si, portanto, mesmo com a mesma frequência, elas não interferem entre si. Apesar disso, é salientado que

neste cenário o uso do mecanismo de Detecção de Atividade da Portadora, o CAD (*Carrier Activity Detection*), pode apresentar resultados errôneos. Este recurso é usado na função Avaliação de Canal Livre, CCA (*Clear Channel Assessment*), de módulos de rádios, permitindo que transmissores consigam detectar se o canal está livre para comunicação, ou então, se devem se manter no modo receptor, proporcionando economia de energia, uma vez que o módulo só é ativo caso detecte uma transmissão. Montanny (2022) acrescentam que o número de bits enviados em um símbolo (tamanho do *chirp*) é igual ao do SF, ou seja, quanto maior o SF, mais tempo de transmissão um mesmo símbolo gasta para ser transmitido.

- **Largura de Banda**: faixa de frequências ocupadas pela transmissão. Diretamente proporcional à velocidade de transmissão e inversamente à sensibilidade devido ao tamanho da porção exposta a ruídos. A Figura 10 ajuda a ilustrar esta propriedade.

- **Taxa de Codificação**: o Coding Rate (CR), conhecido como a taxa de Correção de Erro Encaminhado (FEC, *Forward Error Correction*), fornece proteção contra rajadas de interferências, e quanto maior o CR, mais robusta é a integridade dos dados, por outro lado, menor a taxa de transmissão. O CR, ao contrário dos outros parâmetros citados, não impede a comunicação entre nós se estiver diferente entre pontas. Chang, Onohara e Mizuochi (2010) descrevem que o FEC é um método estruturado basicamente em um encoder que adiciona símbolos derivados de bits dos dados dos pacotes transmitidos para que em caso de falhas, o receptor munido do FEC decoder, possa corrigir e restabelecer a informação original a partir dos dados de backup adicionais enviados codificados.

2.5 HTTP

O HTTP é o protocolo base de todas comunicações com base em navegações na Internet, a rede mundial de computador ou também WWW (*World Wide Web*). É o que explica Wong (2000), onde todas as requisições em mídias como imagens, documentos e formulários funcionam a partir dele. Ainda o qualifica por fornecer um método de fazer diferentes dispositivos comunicarem entre si.

Arquitetado por Tim Berners-Lee nos anos 90, a sigla HTTP vem de ***H****yper* ***T****ext* ***T****ransfer* ***P****rotocol*, apresenta Yannakopoulos (2003), afirmando que a sua versatilidade advém de ser genérico entre os diferentes usos, não guardar estado de operação (*stateless*), e ser orientado a objetos como mídias. Wukkadada et al. (2018) explicam que as comunicações em HTTP são realizadas entre cliente e servidor no modo pedir solicitação e receber uma resposta, no entanto, ele transmite pacotes muito grandes para informações pequenas, levando a sobrecarga na rede (*overhead*), que não é característica desejável em IoT. Além disso, ressalta que em questões de segurança e praticidade, o ideal é que

apenas o cliente seja configurado nos dispositivos da rede por questões de complexidade, desperdício de recursos e brechas para falhas de segurança, apontando que o HTTP é utilizado com frequência nos clientes com a intenção de requisitar dados de um servidor, como exemplo dados meteorológicos. Na Figura 13 tem-se um exemplo de comunicação HTTP.

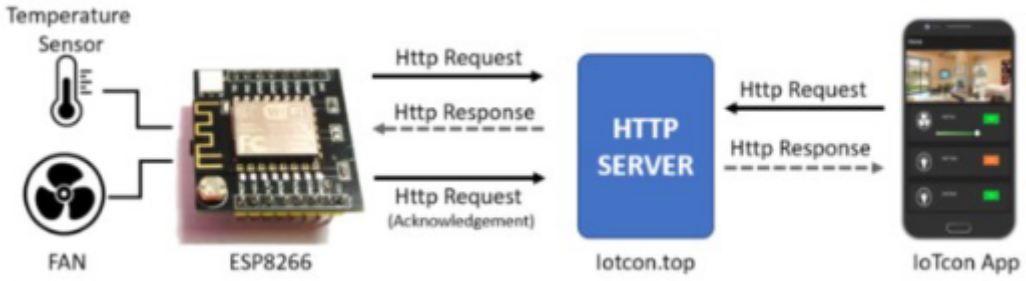

Figura 13 – Requisição HTTP Cliente/Servidor. Fonte: Wukkadada et al. (2018, p. 251).

2.5.1 Solicitações HTTP com URL

As requisições HTTP, esclarece Wong (2000), são realizadas por endereços URL dos servidores (*Uniform Resource Locator*). Segundo Gourley e Totty (2002), as URL são suportadas por outros protocolos como FTP (*File Transfer Protocol*) e o SMTP (*Simple Mail Transfer Protocol*), além de simplificar a navegação online, dispensando a necessidade de comandos avançados para os usuários realizar atividades simples como ler um e-mail.

Wong (2000) relata que no HTTP as URLs possuem o formato semelhante a "`http://example.ora.com:80/`", onde o "*http://*" indica o protocolo utilizado, "*example.ora.com*" o nome do servidor (*hostname*), "*:80*" a porta de operação do serviço (entre 1 a 65535), e por fim o "*/*", o caminho (*path*) para alcançar os recursos. Após uma solicitação, o servidor recebe a requisição associada ao caminho, e de acordo com processamento interno, devolve uma resposta ao pedido junto com o código padronizado de status da operação.

Gourley e Totty (2002) complementam, indicando que o *path* pode ser particionado, e que certas solicitações necessitam que os usuários enviem informações especiais de configuração. A passagem pode ser realizada ao final da URL via *parâmetros* e *query* em uma estrutura chave/valor como apresentado a seguir:

- **Parâmetros**: separados por "*;*" após o caminho, informa ao servidor a forma correta de processar alguns serviços para que a operação seja realizada como esperado. O exemplo a seguir apresenta a URL com o caminho "*/boxes/index.html*" recebendo respectivamente os parâmetros de chaves "*sale*" e "*graphics*" com valores iguais a "*false*" e "*true*".

  - **Exemplo:**
    `http://www.store.com/boxes/index.html;sale=false;graphics=true`

- **Query**: adicionadas ao fim da URL, marcando o início das chaves/valores com "*?*", agrupando cada uma delas com o operador "*&*". Usadas para enviar parâmetros adicionais de configuração dos recursos solicitados, por exemplo, para escolher a unidade de medida da temperatura. A URL a seguir ilustra o uso de Query com as chaves "*item*", "*color*" e "'*size*" unidas pelo caractere "*&*" recebendo respectivamente os valores "*12731*", "*blue*" e "*large*".

  - **Exemplo:**
    `http://www.store.com/invent?item=12731&color=blue&size=large`

2.5.2 Estrutura das mensagens HTTP

As mensagens pelo protocolo HTTP, apresentam Gourley e Totty (2002), possuem três partes diferentes, a linha inicial (*start line*), cabeçalho (*headers*) e o corpo da mensagem (*body*). Os dois primeiros são compostos basicamente de strings ASCII separadas por quebra de linhas. Já o body é opcional e pode receber tanto strings quanto dados de variados formatos como imagens e arquivos binários, sinalizando o tipo ao servidor durante a requisição pelos atributos "*Content-Type*" e "*Content-Length*". A Figura 14 demonstra um exemplo de estrutura possível dos pacotes HTTP.

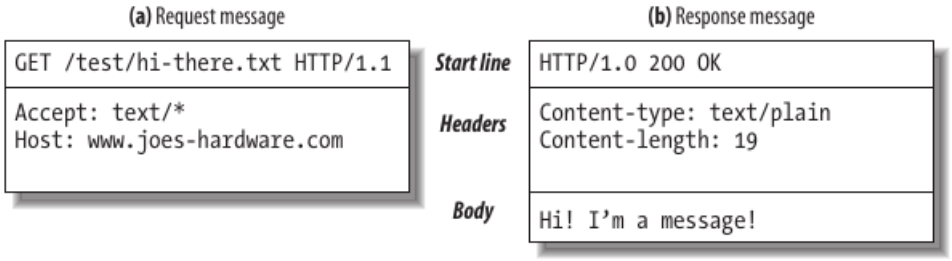

Figura 14 – Partes de uma mensagem HTTP. Fonte: Gourley e Totty (2002, p. 47).

2.5.3 Métodos HTTP

O HTTP disponibiliza métodos para os clientes solicitarem recursos do servidor, esclarecem Gourley e Totty (2002). O servidor os processam de acordo com os parâmetros da URL e estrutura das mensagens (Ver 2.5.1 e 2.5.2), apontando que os métodos mais comuns são:

- **GET**: pede um documento/informação do servidor;
- **HEAD**: pede apenas o cabeçalho (*header*) de um documento/informação do servidor;

- **POST**: envia um documento/informação do servidor (body é obrigatório);
- **PUT**: salva o body (corpo) no servidor (body é obrigatório);
- **TRACE**: rastreamento de solicitação em *loopback* na rede entre cliente e servidor para *debug*;
- **OPTIONS**: retorna os métodos disponíveis para uso no servidor;
- **DELETE**: apaga um documento/informação do servidor.

2.5.4 Códigos de Status

Gourley e Totty (2002) relatam que os códigos de status são utilizados para que o lado cliente consiga interpretar a resposta dos métodos HTTP realizados em um servidor. Eles são padronizados a partir de intervalos de números compostos de três dígitos que indicam o status de retorno (*feedback*) a um pedido solicitado. A seguir apresenta-se resumidamente a distribuição dessas faixas e o que elas sinalizam:

- **(100 – 199)**: informações sobre as requisições;
- **(200 – 299)**: realizada com sucesso. Se destaca dentre elas o 200 "*Status Ok*";
- **(300 – 399)**: avisos de redirecionamento;
- **(400 – 499)**: erro no pedido do cliente. Exemplos são o 403 "*Forbidden*" e o 404 "*Page Not Found*";
- **(500 – 599)**: avisos de erros no servidor ao processar requisição.

2.6 MQTT

O ***M**essage **Q**ueue **T**elemetry **T**ransport* (MQTT), traduzido para Transporte de Telemetria de Fila de Mensagens, é um protocolo desenvolvido por Andy Stanford-Clark da IBM e Arlen Nipper em 1999, esclarecem Yassein et al. (2017). Explicam que o MQTT utiliza um estilo de comunicação máquina para servidor (M2S, *machine-to-server*), servidor para servidor (S2S, *server to server*), máquina para máquina (M2M, *machine-to-machine*) e mecanismos de roteamento (um para muitos, um para um e muito para muitos). Utilizando um servidor central, o *Broker*, estabelece um barramento a partir de uma porta TCP de saída com os dispositivos conectados na rede em um modelo cliente/servidor, onde cada um deles é capaz de se comunicar a partir de comandos *publish/subscribe* (publicar/inscrever), como mostra a Figura 15.

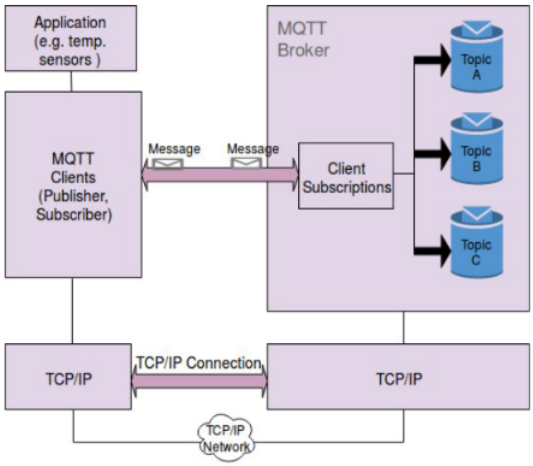

Figura 15 – Arquitetura de comunicação MQTT. Fonte: Yassein et al. (2017, p. 5).

De acordo com Naik (2017), tanto o MQTT quanto o HTTP (Ver 2.5) utilizam o TCP para estabelecer conexão, no entanto ele é mais leve, consumindo menos em questão de banda, memória e energia, mas sendo mais frágil na comparação em nível de segurança. Lampkin et al. (2012) citam que o protocolo é aberto, gratuito e ideal para ambientes com conexões de internet limitada ou com alta latência devido a sua simplicidade, auxiliando a implementação em sistemas de *hardware* limitado, com uma arquitetura capaz de suportar milhares de clientes conectados em um único Broker. Destacam também, dentre outras características, a simplificação no uso por aplicativos e dispositivos, por estes não precisarem conhecer detalhes específicos dos destinatários das mensagens.

2.6.1 Conceitos e Recursos Básicos

O MQTT possui alguns pontos importantes para sua utilização, e que são abordados por Lampkin et al. (2012) conforme listado a seguir:

- **Envio de mensagens**: a comunicação é realizada entre um transmissor (*publisher*, publicador) com identificadores do Broker que recebem o nome de tópicos, propiciando que um ou vários receptores (*subscribers*, assinantes) se inscrevam nestes tópicos e sejam capazes de consumir as mensagens recebidas. Este fluxo de troca de dados entre publicadores e assinantes é ilustrado na Figura 16. O MQTT inclui ainda um recurso avançado que é o de "*Bridge*", fazendo uma ponte entre dois Brokers, compartilhando os tópicos entre eles e propiciando melhora na escalabilidade de aplicações;

- **Sistema de tópicos**: os tópicos definem os diretórios das mensagens que vão ser recebidas pelos dispositivos inscritos neles. Normalmente são usados em uma estrutura de árvore, onde cada nome é separado por um "/" representando níveis

hierárquicos, como “sala01/temperatura”, para criar sub-tópicos. Um recurso presente é o de “*topic strings*”, permitindo o uso de caracteres “curingas” para que os *subscribers* consigam corresponder o padrão de tópicos realizados pelos *publishers.* como mostrado a seguir:

- **Multinível (#)**: permite que o *subscriber* tenha acesso a todos os níveis superiores ao uso do caractere “#” da hierarquia de um tópico, incluindo ela própria. Exemplificando, dispositivos inscritos em “sala01/temperatura/#” receberá dados tanto de “sala01/temperatura” quanto de “sala01/temperatura/sensorA”, “sala01/temperatura/sensorB” e qualquer outra organização derivada deste padrão.
- **Nível único (+)**: usa o caractere “+” como curinga do nível em que ele está presente, capturando tópicos que obedeçam a mesma estrutura considerando qualquer nome como identificador no nível em que o “+” está inserido. Portanto, um cliente inscrito em “sala01/+/sensorA” receberá dados de “sala01/temperatura/sensorA” e “sala01/umidade/sensorA”, mas não de “sala01/temperatura/sensorB” ou “sala02/temperatura/sensorA”;

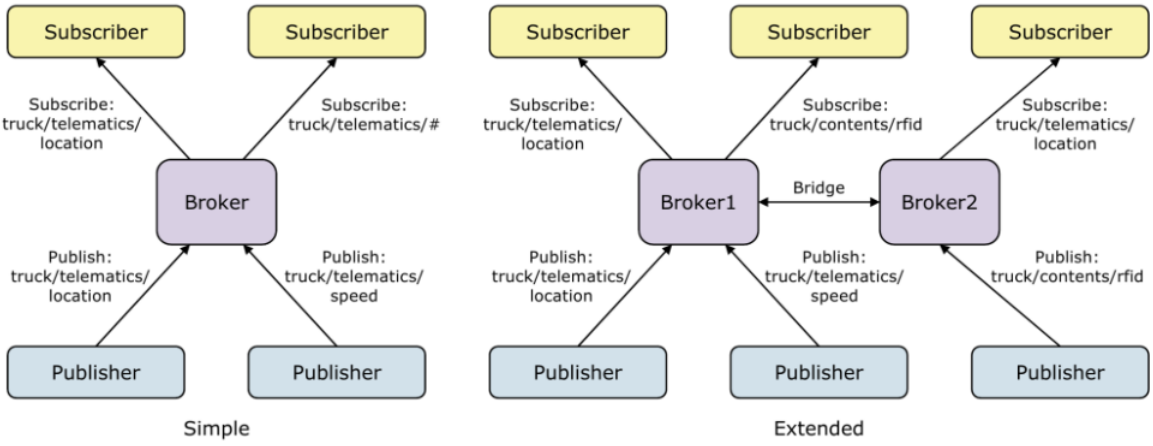

Figura 16 – Estrutura de envio de mensagens em redes MQTT. Fonte: Lampkin et al. (2012, p. 26).

- **ID do cliente**: string de 32 bytes para identificar a conexão de cada dispositivo no Broker, devendo ser único;
- **Quality of Service**: definição de 03 níveis que define a forma que os clientes entregam as mensagens ao Broker (*publishers*). São eles:
  - **QoS = 0, ao menos uma vez entregue**: publicação sem checagem de recebimento, podendo ser perdida em caso de problemas no Broker ou nos clientes MQTT. É o que exerce menor impacto na rede;

  - **QoS = 1, pelo menos uma vez entregue**: garante que a publicação aconteça pelo menos uma vez, recebendo uma confirmação da entrega pelo servidor (*PUBACK, Publish Acknowledged*) como garantia de sucesso no envio. No caso de detecção de falhas, o transmissor reenvia a mensagem após um período de tempo pré-estabelecido, ativando o bit *DUP*, sinalizador de mensagem duplicada no cabeçalho da mensagem. Nesse caso, o Broker republica a mensagem para os assinantes e reenvia o *PUBACK* ao publicador;
  - **QoS = 2, exatamente uma vez entregue**: garante que a mensagem enviada chegue ao servidor apenas uma vez, impedindo que a aplicação receba mensagens duplicadas. É o maior nível de Qualidade de Serviço, mas também o mais custoso em nível de processamento e consumo da rede. Utilizado em aplicações que não podem receber registros duplicados, o QoS 2 possui várias etapas que se iniciam com o cliente publicando no Broker (*PUBLISH*). Este então responde ao cliente que recebeu o conteúdo (*PUBREC*), mas ainda não a entrega aos assinantes. Com a confirmação, o cliente aciona o Broker, solicitando que entregue a mensagem bloqueada para todos cadastrados no tópico alvo (*PUBREL*). Por fim, o Broker realiza os envios e sinaliza o cliente da ação (*PUBCOMP*), finalizando a transmissão e garantindo que ela só seja enviada aos *subscribers* uma única vez.

- **Retenção de mensagens**: recurso que possibilita os clientes ativarem a retenção de mensagens no servidor, assim, novos inscritos nos tópicos recebem sua versão mais recente. Para que funcione, é necessário ao menos o QoS 1, e só é possível reter uma única mensagem por tópico. É um artifício útil para representar o estado atual do tópico para novos *subscribers*;

- **Sessão limpa**: tem efeito apenas para QoS 1 e QoS 2. Basicamente, garante que quando ativo, *cleanSession = true*, o cliente descarte mensagens ainda não recebidas de um tópico ao reconectar ao Broker;

- **Keep alive**: parâmetro utilizado para manter o cliente conectado ao Broker. O *keep alive* é um temporizador que se inicia quando não há mais interações com o servidor. Quando ele estoura, o cliente envia uma única mensagem de *PINGREQ (ping request)* e recebe a de *PINGRESP (ping response)* como resposta. O Broker desconectará uma conexão ativa se este tempo estourar (com um acréscimo de 50%). Cada cliente pode ter um limite específico, o qual é importante se atentar ao tamanho ideal, pois um tempo básico pode gerar muitas requisições na rede, e ainda que essas mensagens sejam pequenas (2 bytes), é indesejado, principalmente se o dispositivo não se comunicar frequentemente com o Broker. Por outro lado, tempos elevados podem fazer com que a comunicação seja perdida por longos períodos sem que isso possa ser identificado.

- **Último desejo e testamento**: configuração pré-estabelecida antes da conexão de um cliente específico que permite o disparo de uma mensagem quando ele é inesperadamente desconectado;

2.7 Raspberry Pi

O Raspberry Pi, esclarecem Crotti et al. (2013), é uma plataforma de desenvolvimento aberta que fornece *hardware* de sistemas computacionais para prototipagem. Desenvolvida pela Raspberry Pi Foundation, tem o intuito de auxiliar o aprendizado da ciência da computação nas instituições de ensino. Os autores destacam que ele é capaz de rodar sistemas operacionais, sendo o Raspbian OS a distribuição GNU/Linux oficial.

Outras características o credencia a forte candidato em aplicações IoT. É o que mostram Maksimović et al. (2014), por ter, além dos aspectos já citados, modelos com disponibilidade de memória RAM, armazenamento expansível, processador ARM com suporte a multiprocessamento (threads) com clock de 700 MHz a 1000 MHz, comunicação Wi-Fi e Bluetooth nativa, capacidade de expansão de portas digitais (GPIO), barramentos digitais como I^2C e SPI e apto para rodar aplicações como servidor. Entretanto, precisa de cartão de memória SD para que o sistema operacional possa ser inicializado. Além disso, é possível programá-lo em várias linguagens de programação, entre elas o C, C++, Java e Python.

Pode ser citado dentro dessa família de embarcados o Raspberry Pi 3 Model B+, apresentado na Figura 17. Conforme a Raspberry Pi Foundation (2023), ele detém como características principais o processador Broadcom BCM2837B0 Cortex-A53 de 64 bits com 1,4 GHz de clock, 1 GB de memória RAM, interface Wi-Fi 2,4 Ghz e 5,8 Ghz, conexão Gigabit Ethernet com suporte a PoE (via HAT), Bluetooth 4.2/BLE, micro SD (Sistema Operacional e armazenamento) e por fim a alimentação de 5 Vdc/2,5 A via micro USB.

Figura 17 – Raspberry PI Modelo 3B+. Fonte: Raspberry Pi Foundation (2023, p. 2).

2.8 RESTful API em IoT

Uma API (*Application Programming Interface*), explicam Maurya et al. (2021), é um conjunto de instruções que permitem pela sua estrutura, a comunicação entre dois diferentes sistemas. A partir dela, podem ser elaboradas formas de acessar determinados recursos de *hardware* e *software* de equipamentos, a exemplo de sensores e atuadores. Um modelo de API é o RESTful, que Garg e Dave (2019) defendem ser uma maneira de padronizar ações de criar, atualizar, ler e apagar informações a partir de comandos em um servidor, que pode ainda, suportar autenticação e proteção contra acessos indevidos, sendo um bom modelo para agrupar e disponibilizar informações de sistemas heterogêneos, uma demanda das redes IoT.

O REST (*Representational State Transfer*), esclarecem Maurya et al. (2021), é uma arquitetura que padroniza o conjunto de protocolos de implementação de servidores Web por meio de uma URI (Identificador de Recursos Universal) com base em métodos HTTP (Ver 2.5), com a URI indicando o endereço de um possível recurso a ser acessado. Estes podem ser tanto informações, status de resposta do servidor ou ainda o formato das respostas recebidas (JSON ou XML). Já o método ou função HTTP para interação, torna possível instruir qual ação o servidor deverá tomar sobre os recursos pedidos. As quatro funções possíveis são basicamente as mesmas suportadas pelo HTTP, se destacando o "GET", "POST", "PUT" e "DELETE", pois a API recebe o termo "RESTful API" caso todas estas citadas estejam implementadas no servidor. Na Figura 18 é ilustrado o processo envolvido nas solicitações de requisições em uma API RESTful.

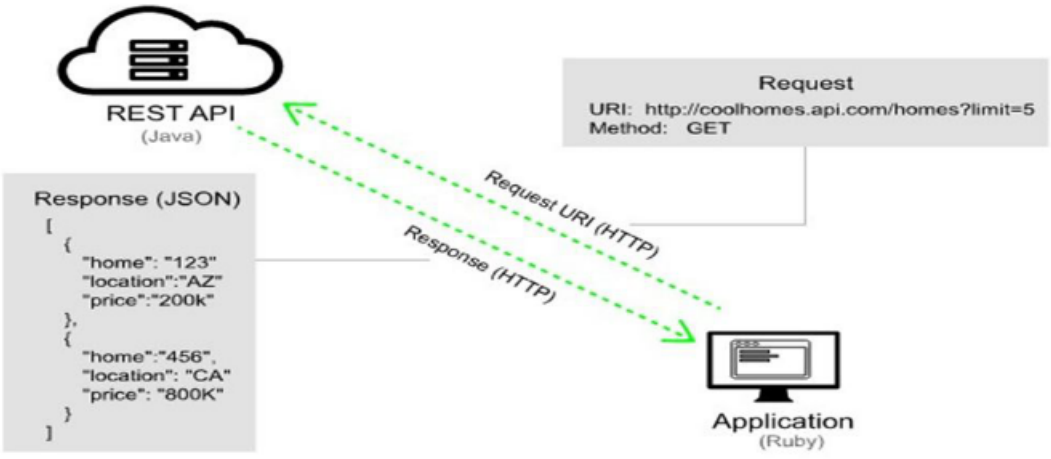

Figura 18 – Aplicação recebendo um objeto JavaScript (JSON) pela requisição HTTP GET em uma API REST. Fonte: Mendes (2021).

3 Desenvolvimento

3.1 Arquitetura do Sistema

O ecossistema heterogêneo da IoT apresenta uma grande variedade de aplicações e tecnologias de comunicação envolvidas (Ver 2.1). Em razão disso, a arquitetura de um Gateway precisa ser pensada ao ponto de integrar no mesmo ambiente dispositivos que sozinhos podem ser incomunicáveis entre si tanto em tecnologia quanto em protocolo de comunicação.

Pensando nisso, a arquitetura do projeto foi inspirada no modelo de Kim, Choi e Rhee (2015) mostrado na Figura 19. Nesta estrutura de um Gateway IoT residencial, tem-se a construção de um sistema com suporte à autoconfiguração para gerenciar dispositivos como atuadores e sensores. Ela utiliza o REST e MQTT na manipulação e monitoramento de parâmetros de status, dados e ações gravados em banco de dados, permitindo a troca de informações com a Internet e nós integrantes da rede. Os elementos que compõe essa plataforma podem ser vistos a seguir:

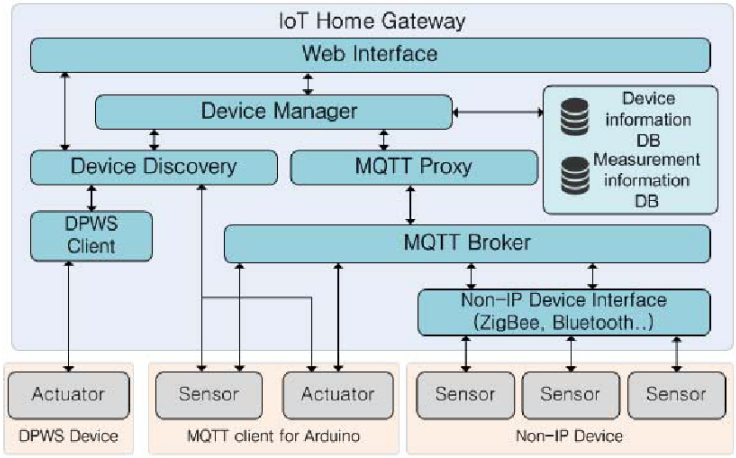

Figura 19 – Arquitetura de um Gateway IoT residencial. Fonte: Adaptado de Kim, Choi e Rhee (2015, p. 13).

- **Gerenciador de Dispositivos (Device Manager):** gerenciador composto de métodos para manipular dispositivos e consultar seus atributos no banco de dados. Com base nele os tópicos MQTT são estabelecidos;
- **Proxy MQTT:** interage com o Broker MQTT, enviando os parâmetros de controle que manipula os dispositivos;

- **Broker MQTT:** o servidor MQTT responsável por manter comunicação do Gateway com os nós;
- **Descoberta de Dispositivos (Device Discovery):** serviço de auto descoberta de nós usando o protocolo DPWS (*Device Profile for Web Services*) e MQTT;
- **Interface Web**: plataforma para manipular e requisitar informações dos dispositivos a partir de API REST;
- **Interface de Dispositivos não IP**: interfaces para comunicar dispositivos de outros protocolos de comunicação diferentes ao MQTT e DPWS.

Os equipamentos, dentro desta proposta de Kim, Choi e Rhee (2015), são identificados por suas características, como visto na Figura 20. Dentre elas se destaca o tipo (atuador ou sensor), nome do dispositivo, localização, comandos de operação (nome e valor correspondente como “LigaTV”:“On”) e a URI de acesso.

Figura 20 – Informações de um dispositivo no Gateway via REST. Fonte: Kim, Choi e Rhee (2015, p. 16).

A arquitetura de Kim, Choi e Rhee (2015) exige que o Gateway tenha conhecimento do tipo e dos recursos que os equipamentos oferecem para que ele possa interagir ativamente enviando os seus comandos de operação armazenados no banco de dados. Essa característica foi evitada por elevar a complexidade do sistema, uma vez que diferentes dispositivos também possuem comandos e requisitos de conexões diferentes. Além disso, na hipótese de nós na rede passarem por modificações ou atualizações, a estrutura de operações já gravadas no Gateway deverá ser corrigida a fim de recuperar o estado operacional, o que pode se mostrar problemático.

A alternativa proposta é fazer com que o concentrador funcione apenas como intermediador transparente na entrega das mensagens. Isso significa que ele deve apresentar duas características, trabalhar como uma ponte de comunicação entre nós e usuários com homogeneidade e desconhecer a natureza das informações trafegadas. Essa arquitetura

está apresentada graficamente na Figura 21. Na busca de atender este propósito, o sistema foi modelado com os seguintes componentes:

- **Interfaces de Rede:** implementa e gerencia as sub-redes derivadas das tecnologias de comunicação utilizadas em integrações IoT. Atualmente disponibilizada versões para o HTTP, MQTT e LoRa;
- **Banco de Dados:** armazenamento de dados como parâmetros de configuração do Gateway, cadastro dos dispositivos e mensagens;
- **Servidor Web:** sistema de gerenciamento para o Administrador a partir de telas de interface Web com acesso restrito com senha, disponibilizando cadastro de dispositivos, configuração das Interfaces de Rede, e demais funcionalidades;
- **Servidor MQTT**: Broker MQTT local para estabelecer conexão de dispositivos MQTT na rede (Ver 2.6) (opcional);
- **API REST:** interface de comunicação usada tanto por nós quanto usuários para requisitar informações do Gateway sobre parâmetros ou mensagens via HTTP.

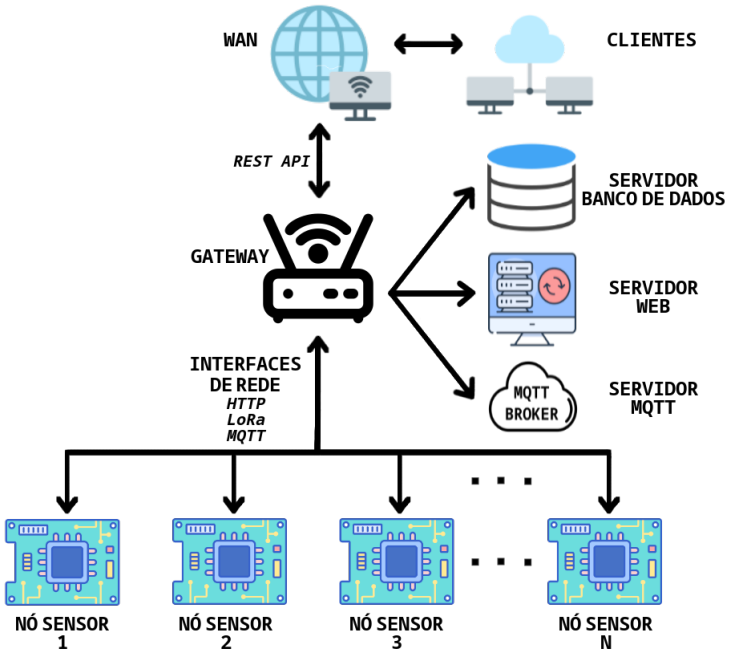

Figura 21 – Arquitetura do Gateway IoT. Fonte: elaboração própria.

Essa arquitetura foi projetada para que o Raspberry Pi 3B+ (Ver 2.7) fosse um sistema computacional independente, ou seja, capaz de ser completamente operacional mesmo em cenários de redes internas sem conexão à Internet ou de acesso a aplicações

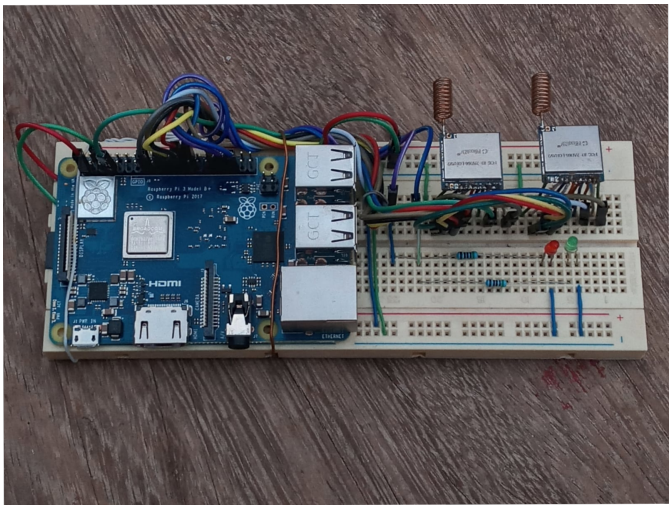

Figura 22 – Montagem de Raspberry Pi 3B+ como Gateway. Fonte: elaboração própria.

terceiras, bastando apenas estar conectado à rede elétrica. A Figura 22 apresenta a montagem do Gateway utilizando o embarcado. Os detalhes desse processo serão abordados mais adiante na Seção 3.6.3.

Com a arquitetura definida em termos de estrutura e *hardware*, é preciso definir as ferramentas de programação de *software* necessárias para criar e integrar os módulos sugeridos. Destas, uma que tem papel importante é o Node.js, escolhida como a linguagem de programação principal na constituição do concentrador.

3.2 Node.js

O Node.js, explicam Tilkov e Vinoski (2010), é uma aplicação para programação em linguagem JavaScript implementada na maior parte em C e C++ com construção visando desempenho e baixo consumo de memória. Enfatiza também que o Node.js se destaca entre outros concorrentes ao não depender exclusivamente de *multithreading* por ser baseada na execução de tarefas assíncronas de entrada e saída associadas a eventos. Uma vantagem em seu uso, aborda Syed (2014), é a versatilidade que ela trás para o trabalho dos desenvolvedores pela possibilidade de usar a linguagem JavaScript tanto na programação de recursos de processamento internos (*backend*) quanto na construção das interfaces de interação com os usuários no lado cliente (*frontend*).

Por essas características, toda a parte do processamento das conexões de rede e gerência dos dispositivos foram escritos utilizando JavaScript como linguagem de programação principal interpretada a partir do Node.js instalado no Raspbian OS (Ver 2.7).

3.3 Servidor Web

O Servidor Web foi preparado no intuito de transformá-lo em uma ferramenta de controle, consulta, gerência e troca de informações e recursos a partir da construção de uma API REST com o protocolo HTTP (Ver 2.8). Para Yeager e McGrath (1996), esses servidores são basicamente compostos de uma plataforma (*hardware* computacional e sistema operacional), do programa em si (*software* do servidor) e da informação a ser disponibilizada aos usuários. Salientam que o trabalho principal dos servidores é fazer com que as requisições dos clientes sejam capazes de ser enviadas para eles, como nos navegadores Web, para poder recebê-las e interpretá-las, devolvendo uma resposta como resultado do pedido enquanto aguardam continuamente por novas solicitações. A Figura 23 retrata o servidor devolvendo a página principal de acesso ao Gateway como resposta ao pedido de acesso ao diretório inicial "/".

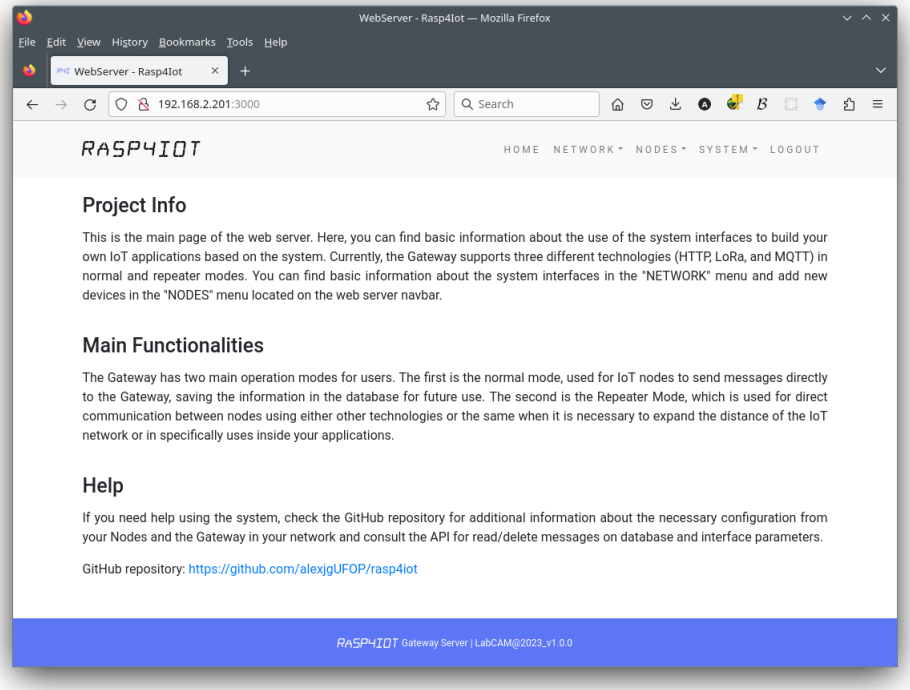

Figura 23 – Servidor Web - Página inicial. Fonte: elaboração própria.

A implementação se deu a partir do Node.js com uso do pacote Express, concedendo um conjunto de ferramentas que auxiliam construções de aplicações Web. A partir dele são preparados os caminhos para estruturação das URL do mesmo modo visto na Seção 2.5.1, fornecendo o meio de interação que viabiliza o disparo de chamadas no sistema em busca de recursos internos abordados com maiores detalhes nas seções seguintes.

3.3.1 Sistema de Autenticação

Parte importante para garantir a segurança das informações e configurações sobre a rede e equipamentos contidas no concentrador é o sistema de acesso por login e senha mostrado na Figura 24. Ele é um dos elementos que promove a vigilância pelo servidor contra tentativas de acessos indevidos a recursos sensíveis, podendo citar entre eles a configuração das Interfaces de Rede, acesso a dados de cadastro dos nós e exclusão de dados.

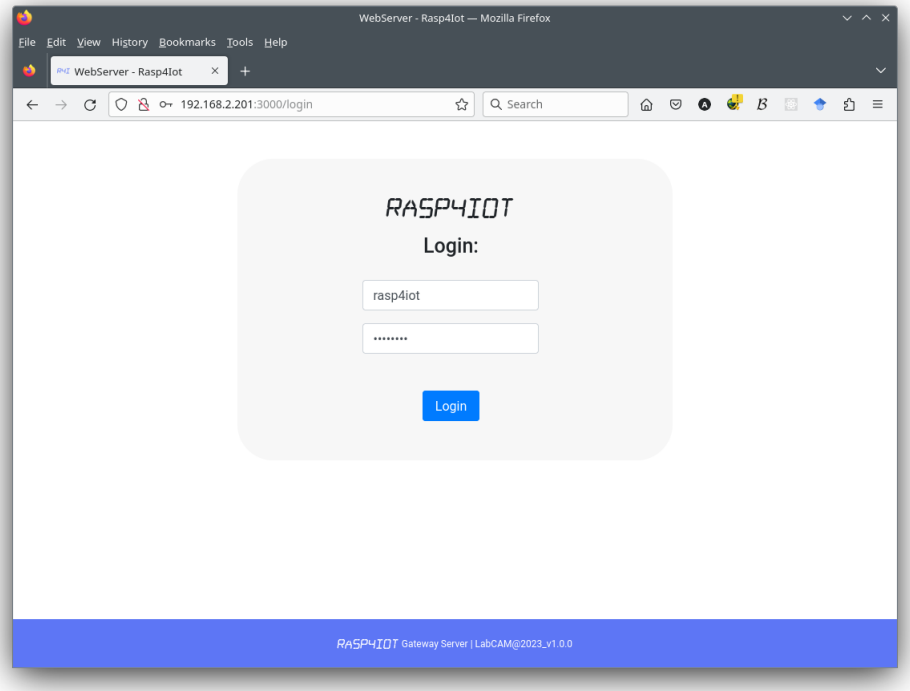

Figura 24 – Sistema de Login. Fonte: elaboração própria.

Na configuração padrão do Gateway, a conta Administrador é definida pelo usuário e senha “rasp4iot”, escolhida por representar o codinome do projeto. Assim, quando um usuário efetua o acesso ao sistema, uma sessão de login ficará aberta em seu navegador até que ele faça o logout na plataforma ou o servidor Web seja reiniciado de alguma forma pelo Gateway. Se ocorrer pedidos de acesso em URL protegidas, o servidor verificará se existe uma sessão ativa. Em caso negativo, a solicitação é bloqueada e o Código de Status 403 (Ver 2.5.4) é entregue como resposta, apontando que esta é uma área restrita. Neste cenário o usuário é automaticamente redirecionado para a página de login.

Por segurança é recomendável trocar a senha padrão do Administrador, uma vez que ela é conhecida e destinada a garantir os primeiros acessos. A substituição é efetuada na página de troca de credenciais disponibilizada e apresentada na Figura 25. Ela recebe e envia a senha antiga e a nova ao servidor, processando a requisição, criptografando e atualizando no banco de dados caso as informações fornecidas sejam plausíveis.

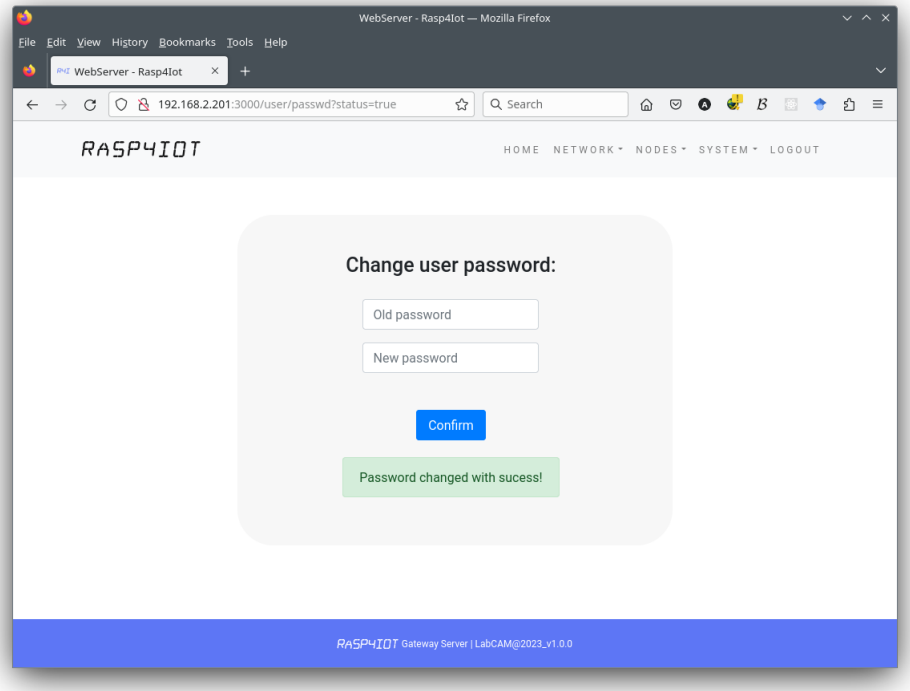

Figura 25 – Troca de credenciais do Sistema. Fonte: elaboração própria.

3.3.2 Cadastro de Dispositivos

Pela inexistência de um sistema de auto-configuração de novos nós como na arquitetura de Kim, Choi e Rhee (2015), é necessário que exista uma plataforma de cadastro dos dispositivos para identificação e habilitação no uso dos recursos do concentrador, como apresentado na Figura 26. Os parâmetros adotados estão listados a seguir:

- **Node Name:** Texto representativo do nome do nó (string);
- **IoT Interface:** interface IoT em uso (disponíveis o HTTP, LoRa e MQTT);
- **Manager Name:** nome do responsável pelo nó;
- **Manager E-Mail:** e-mail do responsável pelo nó;
- **Network Identifier - NID**: endereço de rede gerado aleatoriamente (Ver 2.6);
- **Network API - NAP:** código identificador único de dispositivo gerado aleatoriamente;

O NAP utiliza o UUID versão 4 (*Universally Unique IDentifier*), Identificador Único Universal, também conhecido por RFC 4122. Leach, Mealling e Salz (2005) explicam que ele é um código de 128 bits que dispensa processos de registro, e dentre suas cinco versões, a quarta é gerada utilizando algoritmos do tipo Totalmente Aleatórios ou

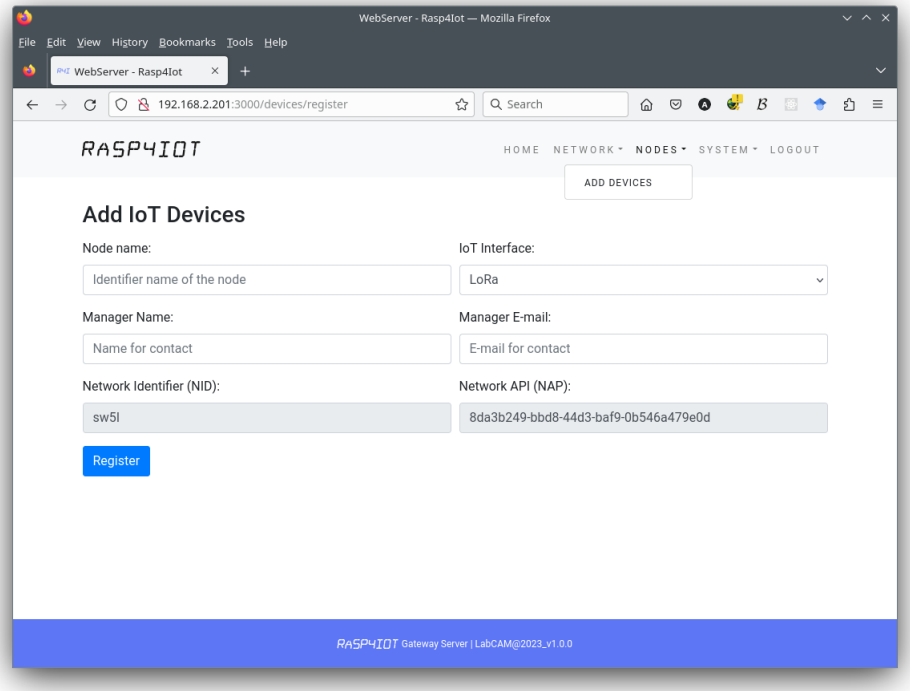

Figura 26 – Cadastro de nós no Gateway. Fonte: elaboração própria.

Pseudo-Aleatórios. O NAP foi combinado com o NID para servir tanto como artifício de controle e manipulação interna dos nós cadastrados quanto também de segurança, restringindo acessos indevidos em mensagens armazenadas no Gateway via API REST. Isso possibilita consultas diretas sobre as mensagens de um nó específico na API, dispensando a autenticação do usuário Administrador nas aplicações dos usuários, bastando que conheçam obrigatoriamente o NID e NAP do alvo.

Finalizado o processo de cadastro de um nó, o servidor redireciona o usuário até a página de gerência da Interface IoT escolhida. Nela, o nó está incluído em uma lista com todos os demais anteriormente cadastrados no sistema, como ilustra a Figura 27.

3.3.3 Gerência das Interfaces

Na implementação e manutenção das redes, em muitas ocasiões pode ser necessário a consulta de informações dos dispositivos e do concentrador, a exemplo, o endereço NID de um equipamento específico ou o identificador NAP para ter acesso às mensagens por uma plataforma qualquer. Além disso, os dados obtidos no Cadastro dos Nós (Ver 3.3.2) podem não ser suficientes para integração de um equipamento, já que as Interfaces possuem configurações e requisitos próprios que precisam ser atendidos, o que podem levar a necessidades de consultas ou ajustes tanto nos nós quanto no próprio Gateway.

Pensando nisso, foi elaborada as páginas de manutenção de cada uma das Interfaces

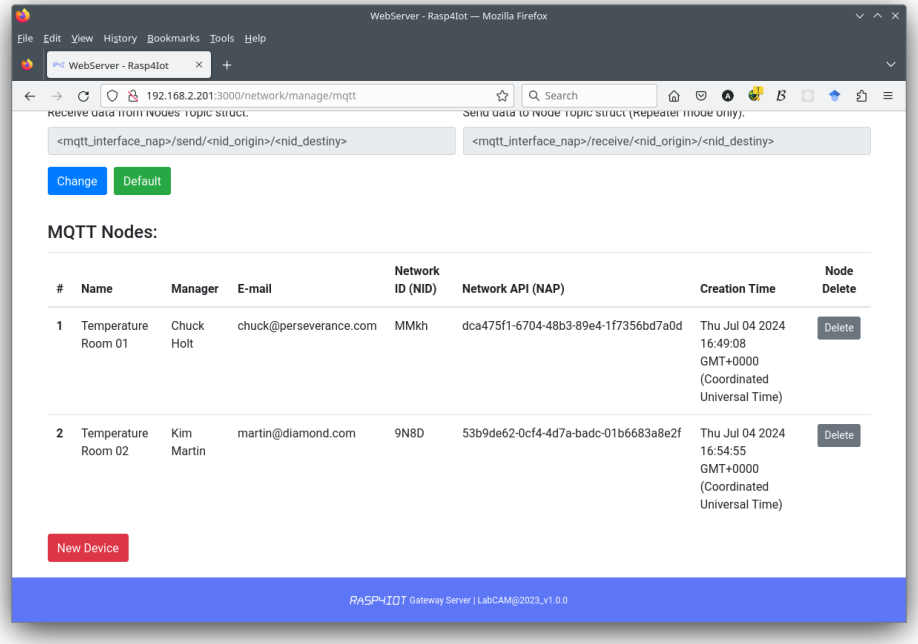

Figura 27 – Lista de nós MQTT cadastrados. Fonte: elaboração própria.

(HTTP, LoRa e MQTT), disponibilizando o acesso aos principais parâmetros usados por elas (Ver 3.6), assim como a lista de todos os nós nelas listados, possibilitando não apenas a visualização deles, mas também a exclusão. As páginas estão representadas nas Figuras 28, 29 e 30.

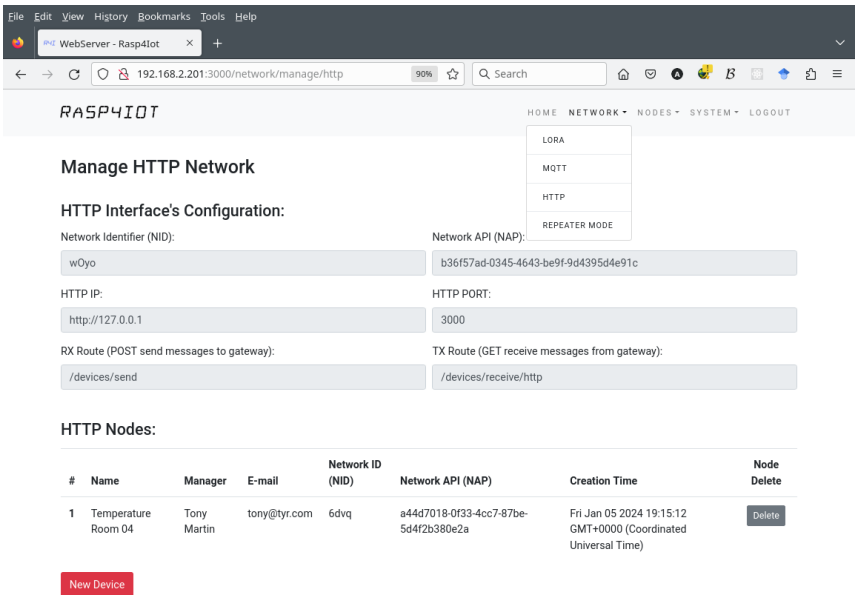

Figura 28 – Página gerência da Interface HTTP. Fonte: elaboração própria.

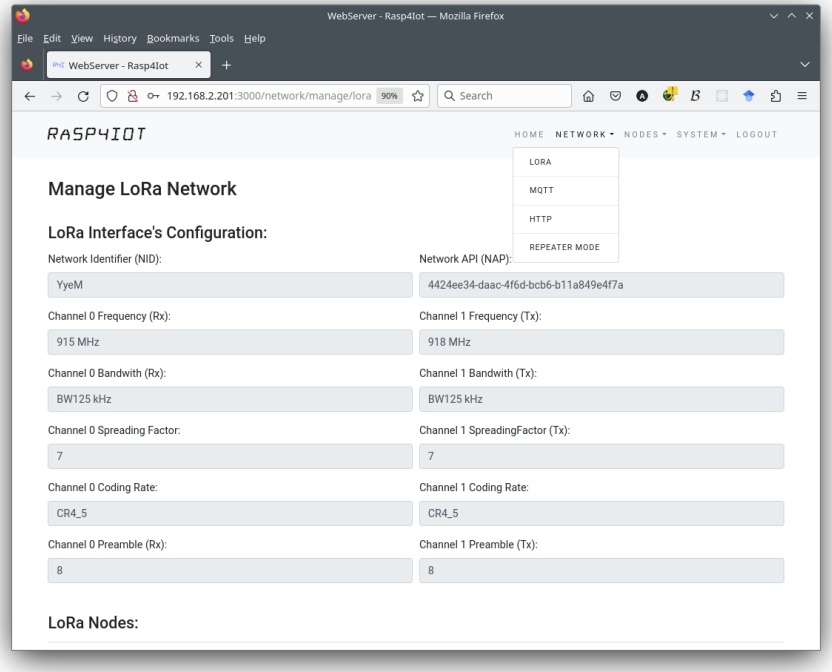

Figura 29 – Página gerência da Interface LoRa. Fonte: elaboração própria.

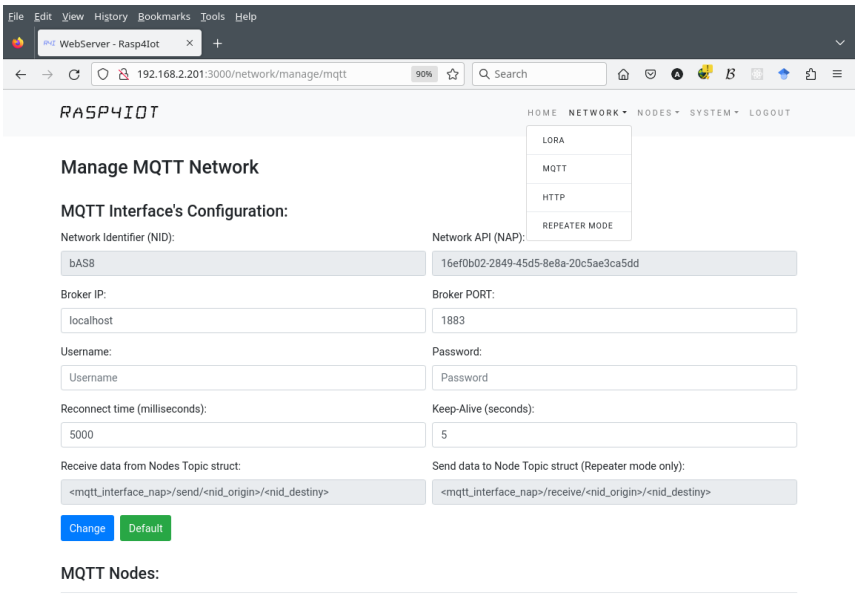

Figura 30 – Página gerência da Interface MQTT. Fonte: elaboração própria.

3.3.4 Roteamento de mensagens

Após o cadastro de um nó no sistema, ele fica ativo, mas só pode enviar mensagens diretamente ao Gateway, para que elas fiquem então armazenadas e disponibilizadas via API REST (Ver 3.3.5), integrando dispositivos sem conexão com a Internet à rede IoT. No entanto, isso não é capaz de tornar a rede completamente heterogênea, uma vez que em certas aplicações um nó precisa se comunicar com outro, entretanto, utilizam diferentes meios de transmissão, o que normalmente os tornam incompatíveis entre si.

Para contornar essa limitação, foi proposto o "*Repeater Mode*", o Modo Repetidor. Ele consiste basicamente em associar o endereço NID do nó emissor com o receptor, como mostra a Figura 31, criando uma rota estática de comunicação. Desta maneira, é possível que um equipamento LoRa envie uma mensagem para outro utilizando MQTT ou HTTP e vice-versa, desde que a rota oposta também seja efetuada.

Quando o Gateway recebe um pacote, é iniciado um processo de verificação de endereçamento que busca checar se tanto o NID de origem quanto de destino são válidos, ou seja, são de dispositivos ou interfaces cadastradas no sistema. Considerando que o concentrador seja o alvo, a mensagem é salva no banco de dados, caso contrário, é analisada se existe uma rota de repetição estabelecida. Em caso afirmativo, os dados recebidos são armazenados em uma fila de mensagens, redirecionando a mensagem para outro nó dentro da rede de acordo com a disponibilidade da interface correspondente de realizar a entrega. Na hipótese de nenhuma dessas condições serem atendidas, a requisição é descartada.

A exigência das rotas, ao invés de simplesmente liberar os nós de se comunicarem livremente entre si, foi para garantir controle, evitando potenciais envios de pacotes inesperados ou indesejáveis para os receptores com origem de emissores desconhecidos dentro do cenário de uma aplicação específica.

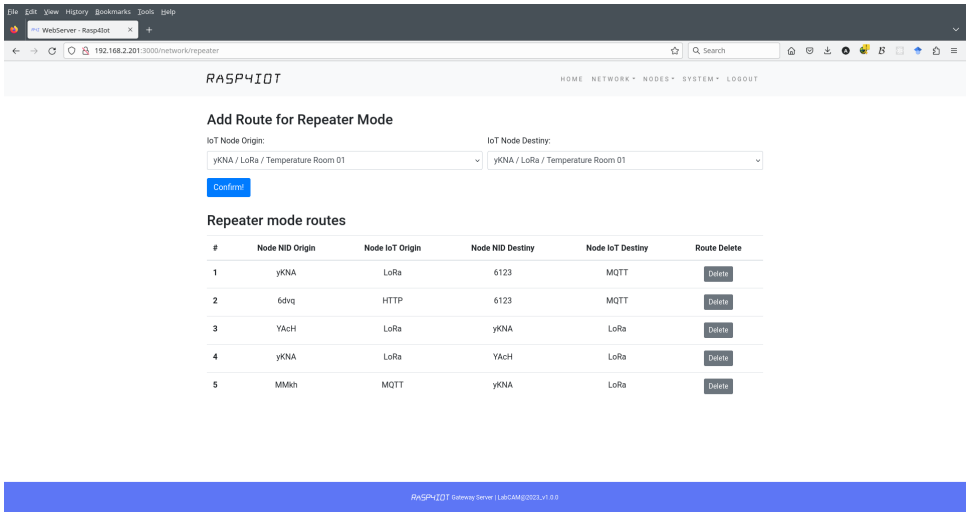

Figura 31 – Página de configuração das rotas de repetição. Fonte: elaboração própria.

Na Figura 32 pode-se ver o *log* de resposta ao tratamento dos pacotes recebidos envolvendo os nós de NID LoRa **YAcH** e **yKNA**, MQTT **MMkh**, o deletado **6123** e a interface LoRa **YyeM** (Figura 29). Nela pode ser visto que o envio entre **YAcH** com **6123** retornou "*Invalid NID destiny!*", considerando a operação como inválida pelo NID **6123** não mais existir, diferente da mesma mensagem que o destinatário é a Interface LoRa (**YyeM**). Logo em seguida, é exibido o registro de atividade envolvendo os nós **MMkh** (MQTT) com **yKNA** (LoRa) acompanhada da indicação de uma rota de repetição encontrada, a mesma presente na quinta linha da lista de rotas na Figura 31.

```
jul 15 07:03:44 rasp4iot node[489]: {
jul 15 07:03:44 rasp4iot node[489]:   interface_nap: '4424ee34-daac-4f6d-bcb6-b11a849e4f7a',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_origin: 'YAcH',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_destiny: 'YyeM',
jul 15 07:03:44 rasp4iot node[489]:   node_message: '52'
jul 15 07:03:44 rasp4iot node[489]: }
jul 15 07:03:44 rasp4iot node[489]: Valid NAP
jul 15 07:03:44 rasp4iot node[489]: Valid NID origin!
jul 15 07:03:44 rasp4iot node[489]: Valid NID destiny!
jul 15 07:03:44 rasp4iot node[489]: Gateway is the destiny, message need to be added on database
jul 15 07:03:44 rasp4iot node[489]: {
jul 15 07:03:44 rasp4iot node[489]:   node_name: 'umidade corredor',
jul 15 07:03:44 rasp4iot node[489]:   node_iot: 'LoRa',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_origin: 'YAcH',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_destiny: 'YyeM',
jul 15 07:03:44 rasp4iot node[489]:   node_message: '52'
jul 15 07:03:44 rasp4iot node[489]: }
jul 15 07:03:44 rasp4iot node[489]: New message receive from YAcH to the gateway!
jul 15 07:03:44 rasp4iot node[489]: User NOT authenticated!
jul 15 07:03:44 rasp4iot node[489]: {
jul 15 07:03:44 rasp4iot node[489]:   interface_nap: '4424ee34-daac-4f6d-bcb6-b11a849e4f7a',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_origin: 'YAcH',
jul 15 07:03:44 rasp4iot node[489]:   node_nid_destiny: '6123',
jul 15 07:03:44 rasp4iot node[489]:   node_message: '52'
jul 15 07:03:44 rasp4iot node[489]: }
jul 15 07:03:44 rasp4iot node[489]: Valid NAP
jul 15 07:03:44 rasp4iot node[489]: Valid NID origin!
jul 15 07:03:44 rasp4iot node[489]: Invalid NID destiny!
jul 15 07:03:44 rasp4iot node[489]: User NOT authenticated!
jul 15 07:03:44 rasp4iot node[489]: LoRa Rx Interface exit data:
jul 15 07:03:44 rasp4iot node[489]: Received:
jul 15 07:03:44 rasp4iot node[489]: YAcHYyeMYyeM52
jul 15 07:03:44 rasp4iot node[489]: New message received in the gateway LoRa interface: YAcHYyeMYyeM52
jul 15 07:03:44 rasp4iot node[489]: Received:
jul 15 07:03:44 rasp4iot node[489]: YAcHYyeM612352
jul 15 07:03:44 rasp4iot node[489]: New message received in the gateway LoRa interface: YAcHYyeM612352
jul 15 07:03:50 rasp4iot node[489]: MQTT topic "16ef0b02-2849-45d5-8e8a-20c5ae3ca5dd/send/MMkh/yKNA" received a new message: 741
jul 15 07:03:50 rasp4iot node[489]: Node MMkh it's a valid MQTT NID Origin!
jul 15 07:03:50 rasp4iot node[489]: Valid NAP
jul 15 07:03:50 rasp4iot node[489]: Valid NID origin!
jul 15 07:03:50 rasp4iot node[489]: Valid NID destiny!
jul 15 07:03:50 rasp4iot node[489]: Exist a repeater mode?
jul 15 07:03:50 rasp4iot node[489]: Repeater Mode found!
jul 15 07:03:50 rasp4iot node[489]: IoT Interface: LoRa
jul 15 07:03:50 rasp4iot node[489]: {
jul 15 07:03:50 rasp4iot node[489]:   node_nid_origin: 'MMkh',
jul 15 07:03:50 rasp4iot node[489]:   node_nid_destiny: 'yKNA',
jul 15 07:03:50 rasp4iot node[489]:   node_message: '741'
jul 15 07:03:50 rasp4iot node[489]: }
jul 15 07:03:50 rasp4iot node[489]: [65B blob data]
jul 15 07:03:50 rasp4iot node[489]: Mensagem na fila: {"node_nid_origin":"MMkh","node_nid_destiny":"yKNA","node_message":"741"}. Código: 200
```

Figura 32 – *Log* do sistema no tratamento de recepção de pacotes

3.3.5 Sistema de consulta de mensagens

Após um nó estar devidamente cadastrado e configurado, torna-se apto para utilizar a Interface IoT. Enviar pacotes ao Gateway, faz com que as mensagens contidas neles sejam armazenadas no banco de dados, podendo ser posteriormente consultadas pelas aplicações independentes dos usuários. A forma escolhida para disponibilizá-los atualmente é apenas por meio da API REST no servidor Web HTTP (Ver 2.5), devido a alta compatibilidade existente com dispositivos que navegam na Internet.

O caminho adicionado à URL para a manipulação das mensagens é o "/devices/-messages". Dois métodos HTTP são suportados, o GET para consultar e DELETE para apagar. Os parâmetros necessários para atender às requisições, iguais para ambos, estão listados a seguir:

1. **Query node_nap:** código NAP do nó alvo. (obrigatório);
2. **Header node_nid:** endereço NID do alvo na rede (obrigatório);
3. **Header all_information:** se "1", retorna todas as mensagens salvas no banco (opcional);
4. **Header message:** busca por uma mensagem específica (opcional);
5. **Header message_id:** busca por uma mensagem com seu o identificador no banco (opcional);
6. **Header data_begin:** data inicial para pesquisa por intervalo de tempo (opcional);
7. **Header data_end:** data final para pesquisa por intervalo de tempo (opcional);
8. **Header last_days:** listagem por quantidade inteira de últimos dias (opcional).

A lista apresenta os parâmetros em ordem crescente de prioridade, o que influencia na escolha do processo de filtragem, uma vez que ela determina que o superior seja o considerado em detrimento aos inferiores. A única exceção é no intervalo de datas, sendo ambos igualmente considerados se enviados. Na Figura 33 tem-se um exemplo de requisição de mensagens de um nó no *software* Postman. Nela pode ser observado a ausência do *node_nap*, mas isso se deve a ele estar presente na aba "*Authorization*" vista na Figura 34.

Na hipótese de preenchimento de dados incompatíveis com os parâmetros, o Código de Status 403 e o *log* de erro indicando a falha é entregue como resposta à consulta. Caso contrário, o servidor devolve os resultados da pesquisa junto ao Código de Status 200 (Ver 2.5.4).

3.4 Servidor MQTT

O Broker MQTT é um dos componentes importantes na arquitetura do Gateway pois ele é parte fundamental para que a Interface MQTT seja funcional. Apesar da Figura 30 mostrar que o próprio Administrador pode escolher qual servidor deseja utilizar, foi adicionado um Broker no Raspberry com a intenção de manter a Interface operacional

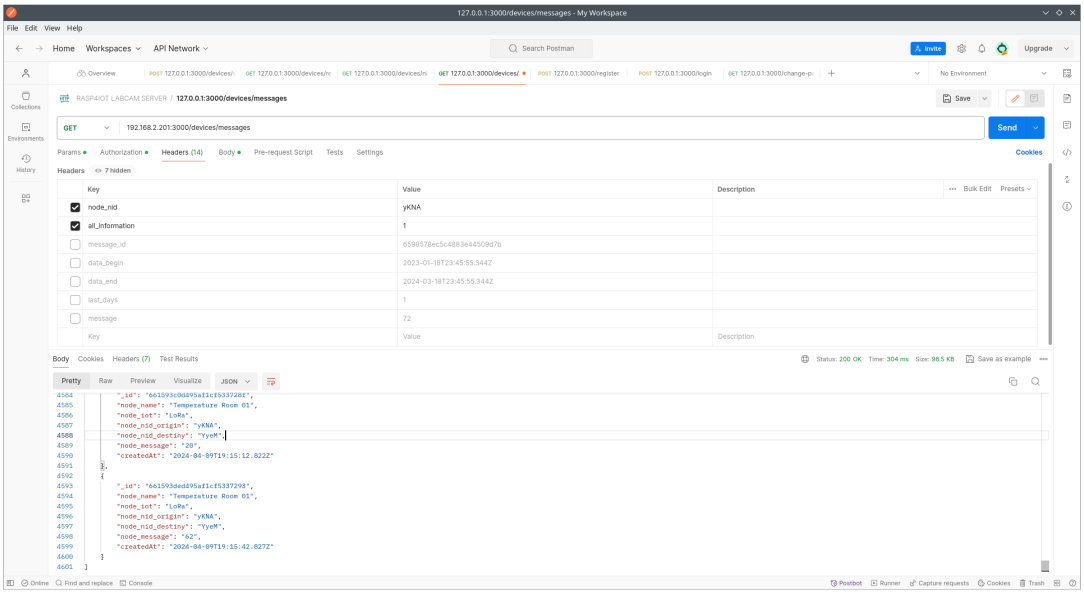

Figura 33 – Requisição de mensagens de um nó no Gateway com o Postman. Fonte: elaboração própria.

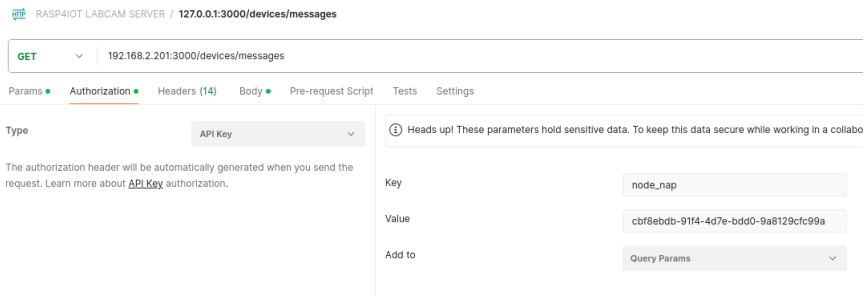

Figura 34 – Adição do código NAP na requisição de mensagens no Postman. Fonte: elaboração própria.

mesmo sem conexão com a Internet. Esta escolha vem da necessidade de cumprir o propósito de manter o concentrador independente de outros serviços e aplicações externas para funcionar.

Para cumprir a tarefa, foi escolhido o Mosquitto da Eclipse Foundation. Light (2017) o apresenta como uma aplicação *Open Source* de implementação do protocolo MQTT para o lado servidor e cliente, preservando as características de baixo consumo de banda e dos recursos computacionais. A Figura 35 mostra o serviço em execução.

3.5 Servidor Banco de Dados

O banco de dados é o responsável pelo gerenciamento de todas as informações, incluindo mensagens, parâmetros de configuração e dados de identificação. A solução

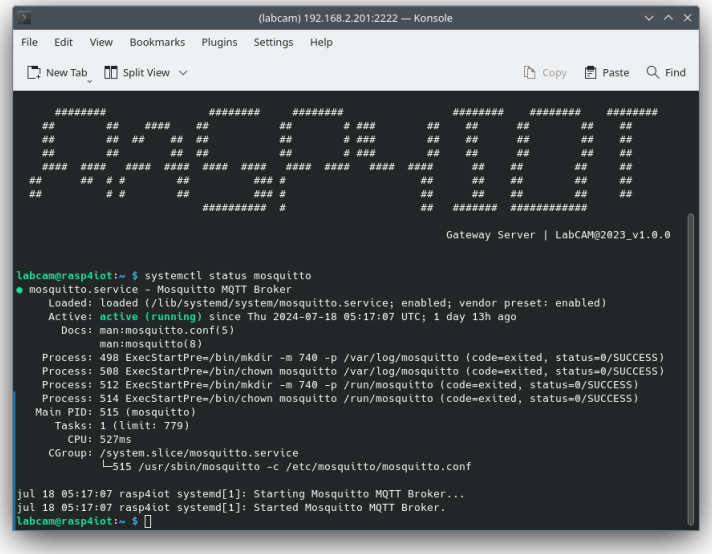

Figura 35 – Broker Eclipse Mosquitto no Raspberry Pi. Fonte: elaboração própria.

selecionada para exercer o papel foi o MongoDB. Bradshaw, Brazil e Chodorow (2019) o definem como um poderoso, flexível e escalável servidor de banco de dados de propósito geral orientado a documentos (objetos), ou seja, não relacional.

Explicam que esta é uma mudança em relação aos modelos relacionais que organizam os dados em linhas de uma tabela. No MongoDB o registro é feito em documentos, o que favorece a escalabilidade. Os documentos são constituídos por modelos de estruturas formadas a partir de níveis hierárquicos personalizáveis compostos de chaves e valores, semelhantes a objetos JavaScript (JSON), que podem ser modificados sem limitações fixas de tipos ou tamanhos. Alterações de atributos dos documentos podem ser feitas a qualquer instante, favorecendo o teste de vários modelos representativos até a escolha de um preferido.

A escolha pelo MongoDB foi influenciada pela flexibilidade na construção do sistema, uma vez que o Gateway tem a proposta de ser heterogêneo, o que por consequência, pode exigir frequentes atualizações para adequar novas tecnologias e requisitos, mas a facilidade de uso também pesa positivamente na seleção. Uma observação é sobre os consumos de *hardware*. Bradshaw, Brazil e Chodorow (2019) citam que o MongoDB utiliza o máximo possível de RAM para cache, e por ser um recurso escasso no Raspberry Pi 3B+ (Ver 2.7), precisou ser checado. A consulta é apresentada na Figura 36, indicando que são reservados 256 MiB da RAM (268435456 bytes) do total (1 GB), o que não compromete a disponibilidade de memória.

Todos os modelos são possíveis representações de documentos associados a uma coleção. Bradshaw, Brazil e Chodorow (2019) relacionam o MongoDB com os bancos rela-

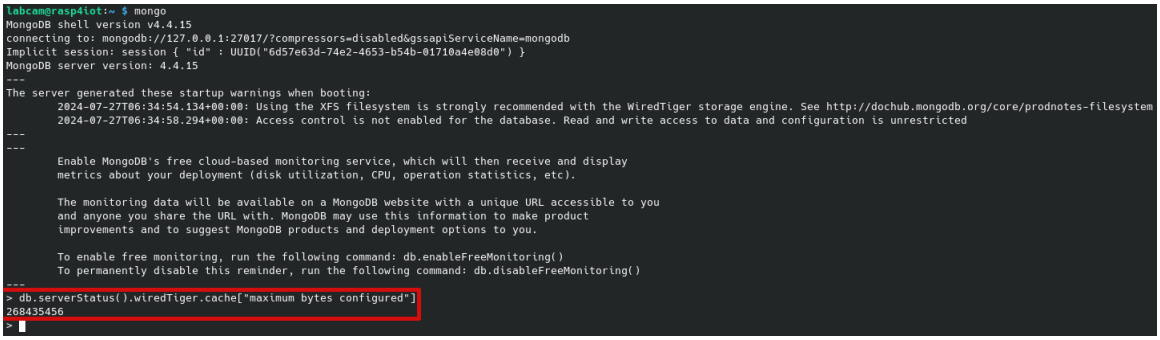

Figura 36 – Memória RAM reservada para cache no MongoDB. Fonte: elaboração própria.

cionais, de forma que as coleções são "tabelas" e documentos as suas "linhas", cada uma com seu próprio identificador único, entretanto, é permitido que as "linhas" se diferenciem sem implicações nas já existentes. O Bloco de Código 3.1 é um exemplo de esquema do modelo escolhido para salvar as mensagens recebidas como documentos da coleção "messages" mostrada na Figura 37.

```
const messageSchema = {
    node_name: String,
    node_iot: String,
    node_nid_origin: String,
    node_nid_destiny: String,
    node_message: String
};
```

Bloco de Código 3.1 – Esquema do modelo de documentos da coleção mensagens

Figura 37 – Mensagens de nós armazenadas no MongoDB. Fonte: elaboração própria.

3.6 Interfaces de Rede

Na construção do projeto, parte importante é estabelecer uma solução para o requisito de heterogeneidade das redes IoT (Ver 2.1), uma vez que cada tecnologia apresenta suas próprias especificidades em relação ao funcionamento. Se influenciando no trabalho de Kim, Choi e Rhee (2015), foram estabelecidas as Interfaces de Rede.

Essas interfaces são os sub-sistemas do Gateway responsáveis diretamente pela gerência das redes, abstraindo o concentrador dos conhecimentos avançados necessários para manipular as técnicas empregadas de comunicação, buscando um comportamento uniforme. Desta forma, o processamento interno consegue tratar diferentes dispositivos como se fossem iguais entre si. Assim sendo, qualquer transmissão é intermediada inevitavelmente pelas interfaces.

Considerando que o sistema aborda o LoRa, MQTT e HTTP, é necessário a implementação para cada uma destas modalidades. No entanto, mesmo que elas sejam capazes por si só de operar suas próprias redes, é necessário que o Gateway exerça controle para identificar quem está apto a utilizar os seus recursos e quais das suas interfaces devem ser acionadas para estabelecer a comunicação.

A solução proposta foi a elaboração de um sistema de cadastro e endereçamento acima das interfaces com o objetivo de criar uma camada de abstração, tornando transparente os processos principais no modo de trabalhar com dispositivos diversos.

3.6.1 Sistema de Endereçamento - NID

O endereçamento é feito a partir de um gerador de Identificador de Rede, também chamado por **NID** (**N**etwork **Id**entifier). Ele é constituído por um código de quatro caracteres alfa-numéricos aleatórios que tem o intuito de distinguir todos os equipamentos que detém meios de conexão, ou seja, tanto os nós quanto cada uma das interfaces possuem um NID único do mesmo formato.

Para que ocorra a comunicação entre elementos, é preciso que sejam conhecidos os endereços dos remetentes, destinatários e principalmente do Gateway. A este último, são atribuídas responsabilidades de escutar os pedidos de troca de mensagens e estabelecer as rotas de trocas de mensagens apenas para os nós em que o NID é conhecido. Isto é, que foram criados pelo próprio concentrador. A Figura 38 é uma amostra de rede típica utilizando o processo descrito.

3.6.2 Interface HTTP

A interface HTTP é uma API REST escrita dentro do Servidor Web com Node.js (Ver 3.3). Essa escolha parte de que o Servidor Web já utiliza o HTTP, e por esse motivo

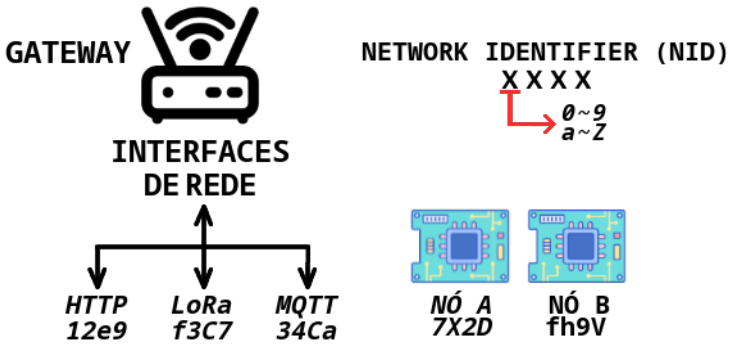

Figura 38 – Exemplo típico de endereçamento NID. Fonte: elaboração própria.

não seria justificável adicionar mais uma instância de Servidor HTTP apenas para a Interface, o que aumentaria desnecessariamente a complexidade de programação e uso dos recursos de *hardware*.

Duas rotas no servidor HTTP são dedicadas para a Interface atender requisições dos nós HTTP. A primeira é "/devices/send", onde os nós enviam mensagens com o método POST. A segunda é o "/devices/receive/:iot", usada para que os nós consultem com o método GET se existe alguma mensagem armazenada na qual sejam o destinatário. Existe uma limitação em ambas solicitações, os pedidos devem partir dos nós.

Na Seção 2.5 é dito que as requisições HTTP são realizadas entre cliente e servidor, assim como esperado das aplicações TCP/IP. Portanto, não é possível que o Gateway notifique os nós via HTTP, a menos que eles também implementem o lado servidor do protocolo. Esse contexto conduz novamente a elevação geral de complexidade, e consequentemente foi desconsiderado. Isso adiciona uma fragilidade operacional que deve ser ponderada, uma vez que qualquer nó desta modalidade que não cheque com frequência por novas mensagens, pode aumentar as chances de perder mensagens mais velhas por descarte permanente no controle de memória RAM.

Uma alternativa viável seria utilizar adicionalmente Sockets para disparar a sinalização de eventos (Ver 2.3). Nessa hipótese, o cliente obrigatoriamente inicia e mantém a conexão aberta com o servidor, possibilitando comunicação *full-duplex*. Mesmo sanando a limitação, ter os nós ininterruptamente ativos torna-se uma barreira, já que a disponibilidade elétrica muitas vezes é um desafio. Em vista disto, essa abordagem também foi rejeitada.

A Figura 39 mostra como os nós HTTP se comunicam na rede, assim como todos os parâmetros que devem ser enviados para que os pedidos sejam aceitos pelo Gateway. É importante lembrar que a comunicação entre nós precisa da adição da rota de redirecionamento (Ver 3.3.4).

Se o código NAP da interface estiver incorreto, a requisição por segurança é bloqueada, impedindo acessos não autorizados e devolvendo o Código de Status 403. Porém, se válida, os dados são encaminhados ao processamento interno, onde são checados a procedência dos endereços NID de origem e destino para de fato executar a operação. Nessa etapa, independentemente das informações enviadas serem válidas ou não, o Código de Status 200 é enviado a fim de indicar sucesso na requisição HTTP apenas, o que não significa que a comunicação será atendida. Fica com o Administrador do nó a responsabilidade de se atentar ao correto uso da API, podendo verificar os *logs* no console do Gateway como indicado na Figura 32 da Seção 3.3.4, se necessário. A página de gerência da interface HTTP (Ver 3.3.3) também pode ser usada em consultas rápidas, contendo todas as informações básicas essenciais.

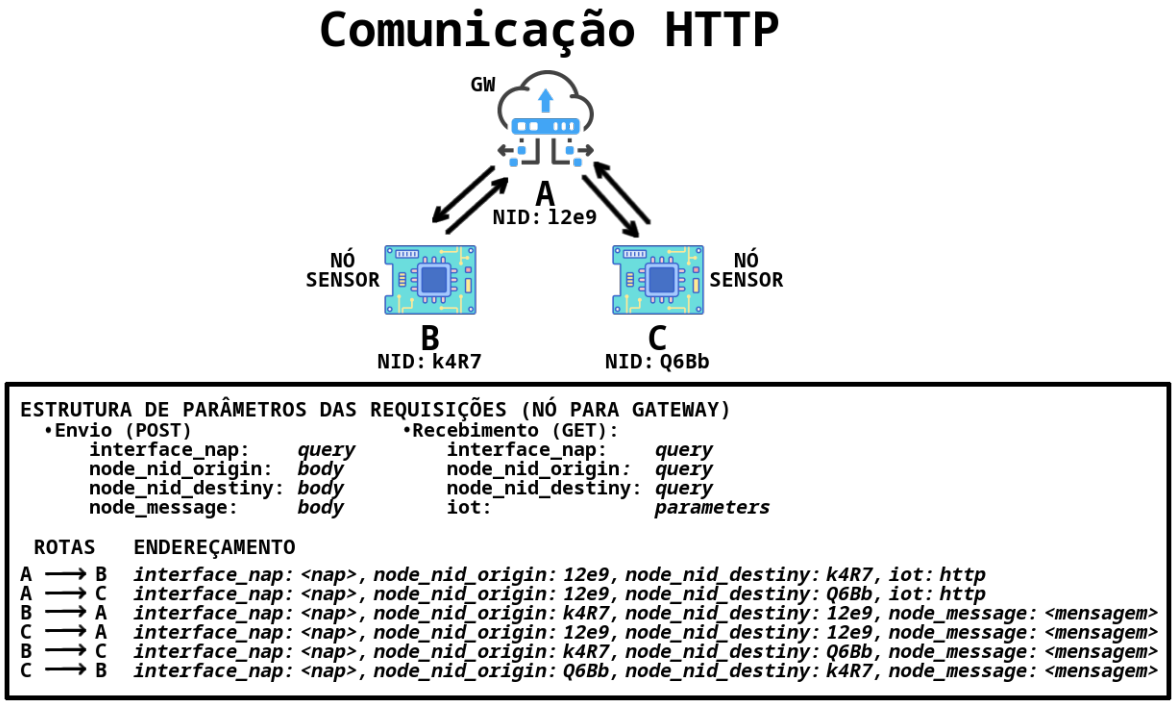

Figura 39 – Comunicação HTTP entre Nós e Gateway. Fonte: elaboração própria.

3.6.3 Interface LoRa

Diferente das demais interfaces de rede, a LoRa é a única que demanda módulos de conexão externos ao Raspberry. Para cumprir este papel foi utilizado um par de rádios 2AD66-LORAV2 da NiceRF Wireless Technology LTD (2017). Segundo o fabricante, eles incorporam no produto o rádio SX1276 da SemTech Corporation, tendo os seus principais atributos de operação mostrados na Tabela 1.

Na Figura 40 observa-se um bloco simplificado dos rádios da família SX127x. De acordo com o *datasheet* da Semtech Corporation (2020), no modo receptor os sinais de entrada passam inicialmente por amplificadores de baixo ruído, ou LNA (*Low Noise Am-*

Tabela 1 – Características do 2AD66-LORAV2 (SX1276) (Adaptado). Fontes: NiceRF Wireless Technology LTD (2017) e Semtech Corporation (2020).

Parâmetro	Mínimo	Típico	Máximo	Unidade	Condição
Limites de operação					
Tensão	1.8	3.3	3.7	V	
Temperatura	-40		85	°C	
Consumo de corrente					
Modo RX		10.8		mA	
Modo TX		120		mA	Potência 20 dBm
		87		mA	Potência 17 dBm
		29		mA	Potência 13 dBm
		20		mA	Potência 7 dBm
Sleep Mode		< 0.2		μA	
Parâmetros de Rádio Frequência (RF)					
Range de frequência	800	868	900	MHz	Freq. Central 868MHz
	900	915	1000	MHz	Freq. Central 915 MHz
Largura de banda	137		1020	MHz	
Spreading Factor	6		12		
Taxa de modulação	1.2		300	Kbps	FSK
	0.018		37.5	Kbps	LoRa
Potência de saída	-1		20	dBm	
Sensibilidade		-123		dBm	FSK
		-139		dBm	LoRa

plifier). Em seguida, são encaminhados para a etapa do *mixer*, convertendo o sinal em componentes de fase e quadratura da frequência portadora para a intermediária (FI). Neste estágio, o sinal digitalizado por ADCs (Conversores Analógicos-Digitais) é processado e demodulado pelo modem LoRa ou FSK/OOK, esta última não explorada no trabalho. Este estágio é responsável por converter e processar o sinal de radiofrequência (RF) e transformá-lo em informação a ser disponibilizada nos registradores do módulo acessados via barramento SPI.

As frequências de transmissão e recepção são alcançadas por dois diferentes sintetizadores (PLL) que usam o cristal oscilador para gerar as linhas de classe baixa (até 525 MHz) e alta (acima de 779 MHz). Na transmissão, o sinal percorre o caminho inverso e é entregue para três possíveis amplificadores de potência. Eles são o RFO_LF e RFO_HF para a classe baixa e alta das frequência de operação com potência de sinal até 14 dBm, e o PA_BOOST para 20 dBm no máximo dentre todas frequências suportadas. Outra observação de ambos fabricantes é sobre a impedância de entrada da antena, que precisa ser de 50 Ω.

Toda a transferência de dados é realizada em pacotes com a estrutura apresentada na Figura 41, que contém os conceitos de Taxa de Codificação (CR - *Coding Rate*) e Fator de Espalhamento (SF - *Spreading Factor*) discutidos na Seção 2.4.2.

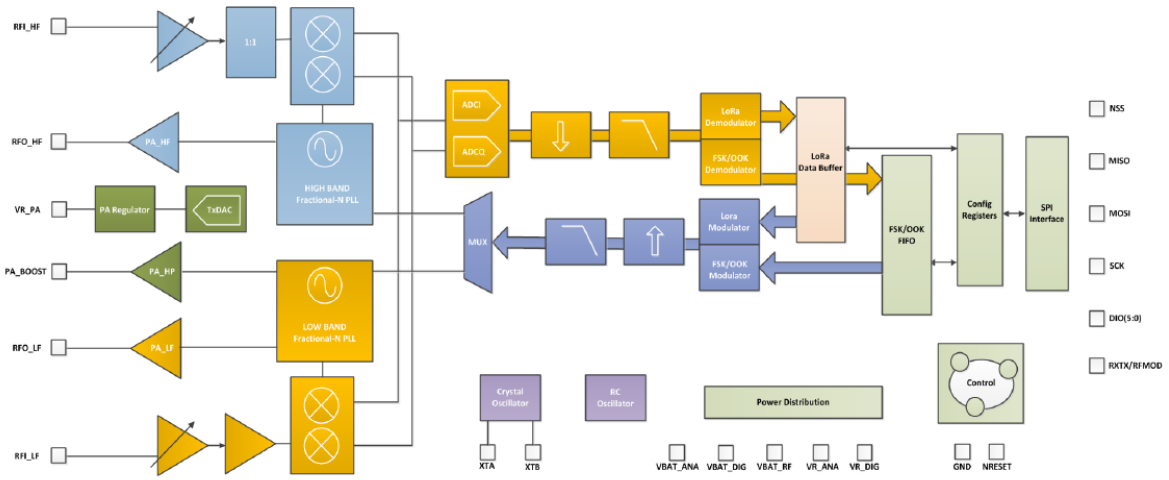

Figura 40 – Bloco esquemático da família SX127x. Fonte: Semtech Corporation (2020, p. 23).

De acordo com a Semtech Corporation (2020), o preâmbulo (*preamble*) é parte do pacote responsável pela sincronização entre transmissor e receptor. Por consequência, precisa ser igual em ambos os lados. O tamanho padrão é 12 símbolos, mas ajustável entre 6 e 65535 símbolos. A redução do preâmbulo potencialmente otimiza aplicações de elevada taxa de troca de dados.

O cabeçalho (*header*) apresenta informações sobre a mensagem (*payload*). Semtech Corporation (2020) destaca dois tipos. O primeiro é o explícito, contendo o tamanho do *payload*, o tamanho do CR para possíveis correções pelo FEC, e opcionalmente o CRC (*Cyclic Redundancy Check*, Verificação de Redundância Cíclica) usado na análise de integralidade e descarte de mensagens corrompidas. O segundo é o implícito (omitido), obrigatório para o SF6, nele cabe ao usuário fornecer as informações de tamanho da mensagem, CR e se desejável o CRC em ambas pontas do enlace.

No campo de *payload* é reservado o espaço que contém a mensagem e o CRC. Conforme Semtech Corporation (2020), essa informação é salva em uma fila FIFO (*First Input First Output*, Primeiro a Entrar Primeiro a Sair) de 256 bytes dividido na metade entre transmissão e recepção, mas com suporte ao ajuste de particionamento entre as parcelas.

Para a construção e testes da Interface LoRa foram adotados os parâmetros ilustrados na Figura 29. Estes valores são os iniciais dos módulos (com exceção da frequência), mantidos no intuito de inicialmente facilitar a adição de novos nós sem que os usuários precisem realizar alterações avançadas.

A escolha pelo par de rádios decorre da característica *Half Duplex* apresentada pela Semtech Corporation (2020), ou seja, podem atuar como transmissores e receptores, mas não ao mesmo tempo. Portanto, a seleção visa evitar problemas de sincronização,

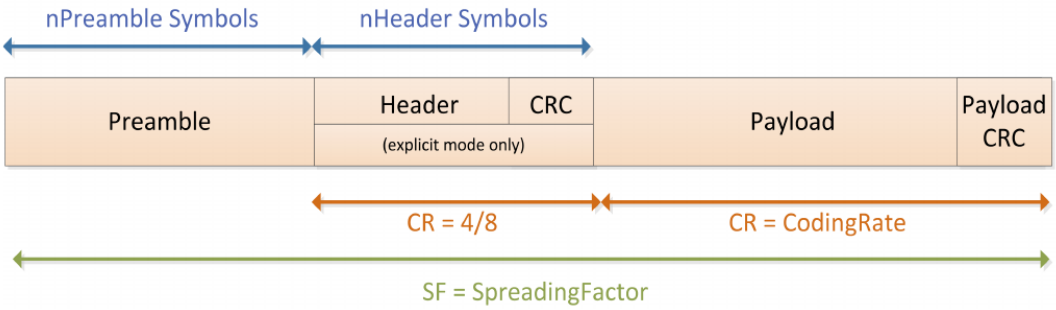

Figura 41 – Cabeçalho de pacote LoRa. Fonte: Semtech Corporation (2020, p. 29).

quedas forçadas do *link* e perdas de pacotes na troca de modos pela inclusão de canais exclusivos para transmissão e recepção, sem causar interferência entre si ao considerar a diferença de frequência mínima entre eles propositalmente maior que a largura de banda.

Semelhante ao discutido na Seção 2.2 a respeito do modelo OSI, o LoRa possui características de protocolo de camada física. Conforme Montanny (2022), ele se comporta como a camada física do protocolo LoRaWAN, usado para criar redes de dispositivos LoRa a partir de uma estrutura enviada dentro do *payload* do pacote LoRa.

Para efeito de simplificação, a interface foi criada utilizando apenas o LoRa. Se inspirando no modelo OSI, foi construído um protocolo formado em texto puro no *payload* para que os nós da rede pudessem se comunicar entre si, como mostra a Figura 42, reunindo o Gateway (GW) mais dois nós (A e B). O padrão busca características das sete camadas do OSI (Ver 2.2). Na Física com o LoRa, o Enlace como a própria Interface validando a procedência, destino e detecção de erro sobre as mensagens, a de Rede, Transporte, Sessão e Apresentação unidas por processamento interno no Gateway comum a todas as outras tecnologias de comunicação, fornecendo recursos de endereçamento, roteamento, métodos de envio entre nós e preparação das informações para entrega de apresentação. Por fim, a camada de Aplicação não é de fato elaborada, considerando-a como comportamento individual que os próprios usuários programam seus nós.

O código da Interface para manipular os módulos foi escrito em linguagem Python utilizando a biblioteca pyLoRa 0.3.1 desenvolvida por Silva (2019). Cada rádio tem seu próprio *script*, ou seja, para o receptor (Rx) e transmissor (Tx). Eles são automaticamente iniciados pelo processo principal do Gateway no Node.js (Ver 3.2), onde é realizada a troca de informações e parâmetros de configuração para inicialização correta da rede LoRa conforme estabelecido na página de gerência da Interface no Servidor Web (Ver 3.3.3).

Considerando o cenário do receptor, o recebimento de um pacote LoRa é detectado pela checagem do registrador da fila de mensagens do rádio. Assim, o *payload* detectado deve conter a estrutura proposta do protocolo no modo de **transmissão** apresentado na Figura 42. É exigido que o *payload* siga o NID na sequência de origem, Gateway

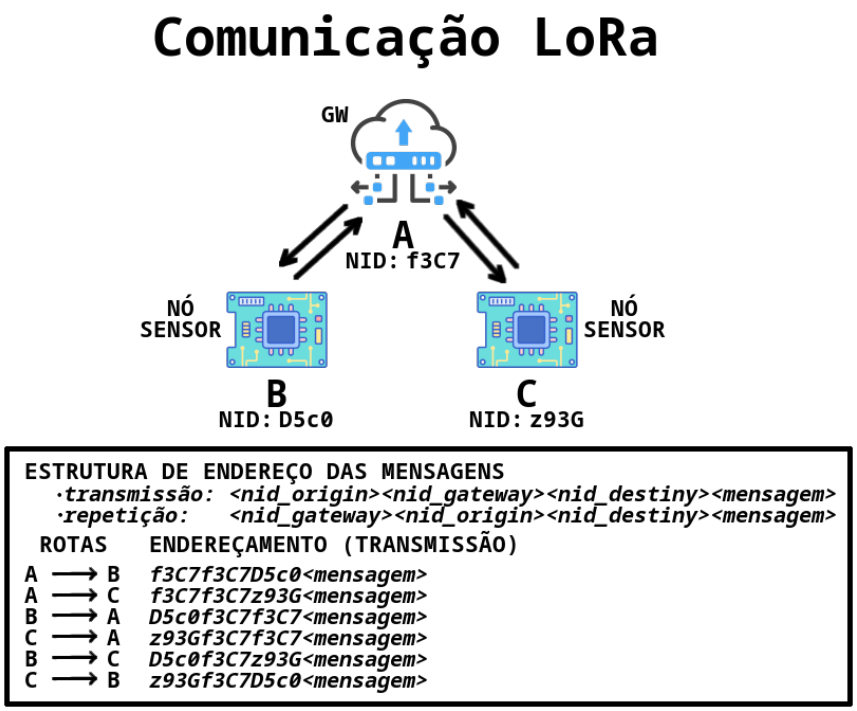

Figura 42 – Comunicação LoRa entre Nós e Gateway. Fonte: elaboração própria.

e destinatário, compondo 12 bytes mais a mensagem desejada. Se ao invés de um nó qualquer, o destino é o Gateway, seu NID deve se repetir também como destinatário, mantendo fixa a faixa do *payload* reservada para endereços. O *script* do rádio Rx em execução analisa se o protocolo é obedecido, em seguida, confere se o NID do Gateway corresponde com o do pacote. Se a condição é atendida, a mensagem é encaminhada para o processo interno de roteamento (Ver 3.3.4), caso contrário ela é descartada.

Para o transmissor, o Gateway consulta se existem requisições de encaminhamentos destinados à Interface LoRa. Detectando um pedido pendente, o concentrador encaminha os endereços (origem e destino) e a mensagem para o *script* responsável pelo rádio Tx em execução. Nele, o *payload* é organizado, gravado no registrador da fila e posteriormente enviado no pacote LoRa do módulo. Uma observação é a variação do endereçamento em duas situações relacionadas com a origem do pedido. A primeira é quando a procedência é do próprio Gateway, obedecendo o modelo de **transmissão**. O segundo é partindo de um nó (independente da tecnologia empregada), seguindo o padrão de **repetição**, onde os NID do Gateway e da origem são trocados entre si de posição. Este artifício é utilizado para que o rádio Rx detecte e descarte o pacote do módulo Tx, evitando *loop* infinito na rede.

Os nós LoRa devem implementar a estrutura de **transmissão** do protocolo no envio de mensagens e o de **repetição** na captura. Lembrando que é determinado que nós só se comunicam entre si utilizando o Gateway ao configurar as rotas de repetição (Ver

3.3.4).

Os dois rádios usados na Interface são conectados ao Raspberry Pi por meio do barramento SPI. A Figura 43 mostra o diagrama de conexão deles no embarcado como o U1 (LoRa Rx) e U2 (LoRa Tx), assim como os LED indicadores de transferência, piscando como sinalizadores. O Processo de montagem está apresentado na Figura 44.

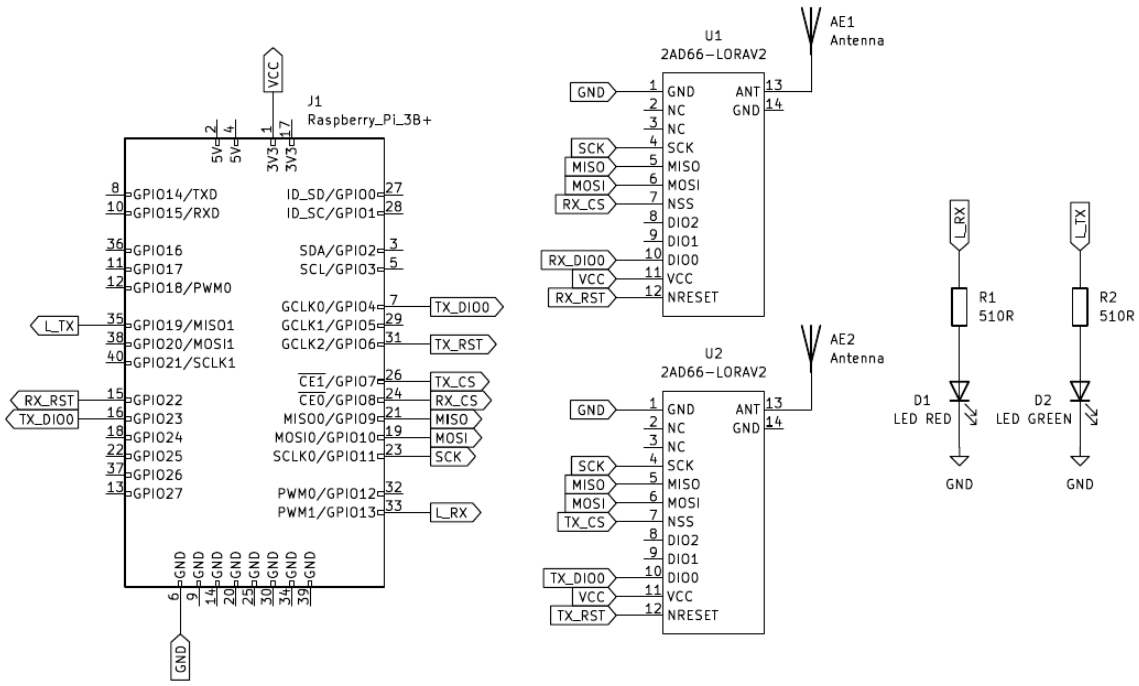

Figura 43 – Circuito esquemático de conexão dos módulos LoRa. Fonte: elaboração própria.

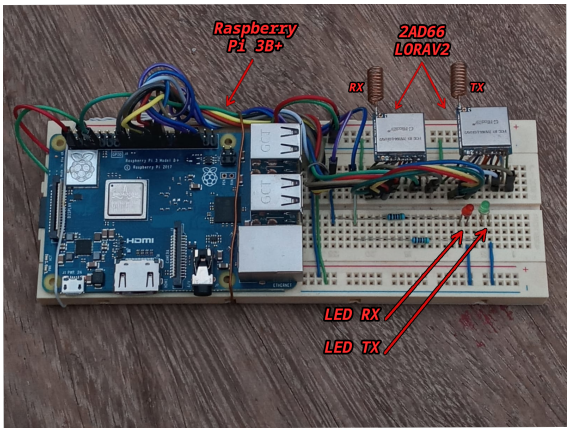

Figura 44 – Montagem dos rádios LoRa no Raspberry Pi. Fonte: elaboração própria.

3.6.4 Interface MQTT

Escrita em Node.js, a Interface MQTT é uma implementação cliente do protocolo, diante disso, necessitando do Broker MQTT como abordado na Seção 3.4. Quando conec-

tada ao Broker, realiza o processo de gerência de tópicos para que o Gateway consiga ter controle sobre os nós que precisam utilizar seus recursos a partir desta interface.

Os principais parâmetros necessários para configura-lá são o endereço do servidor, a porta TCP/IP, login (opcional), ID do cliente, tempo de reconexão e o *keep alive*. Todos eles abordados na Seção 2.6 e editáveis pela página de gerência da Interface MQTT exibida na Figura 30, com exceção do ID do Cliente, gerado aleatoriamente em tempo de execução. Seus valores, assim como de todas as outras interfaces, por padrão usam valores pré-estabelecidos carregados do banco de dados.

Sabendo que a troca de informações no MQTT ocorrem por publicações e inscrições de clientes em tópicos do Broker, é preciso que os integrantes da rede saibam de antemão quais são relevantes. Com esta finalidade, foram estabelecidos os modelos para tópicos de recepção e transmissão envolvendo qualquer endereço NID do sistema. Este esquema está exposto na Figura 45, mostrando três importantes dados, o NAP da interface, os comandos para envio (*send*) e recepção (*receive*) e os endereços NID da origem e destino.

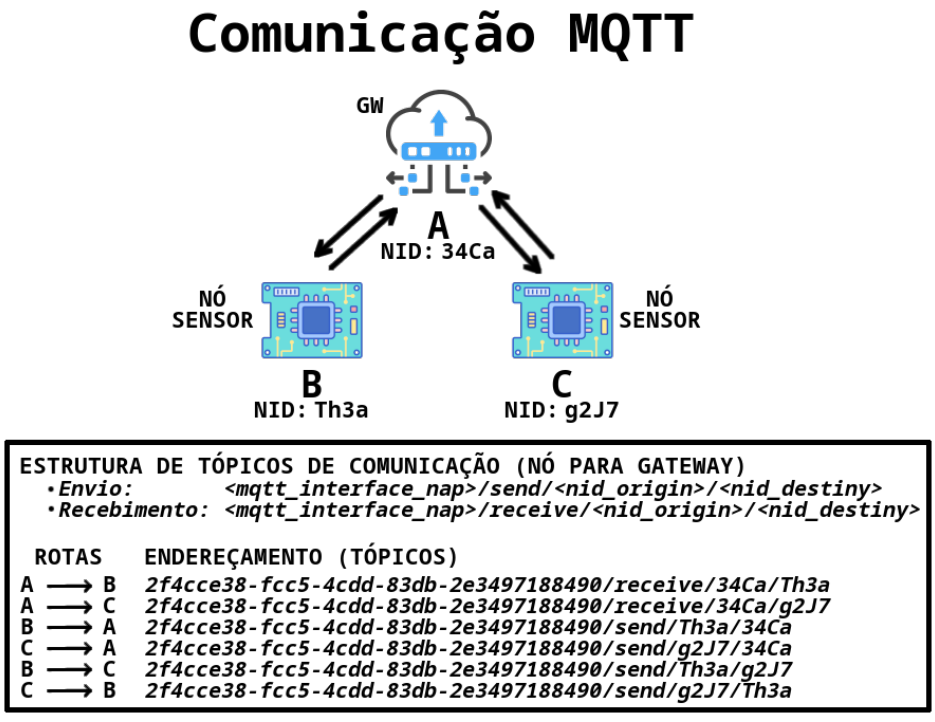

Figura 45 – Comunicação MQTT entre Nós e Gateway. Fonte: elaboração própria.

Apesar da adição de nós gerarem maiores diversidades de tópicos, na prática o cliente MQTT precisa apenas do curinga nível único "+", dispensando lógicas complexas de inscrição pela substituição do padrão a seguir:

- **Recepção (Gateway recebe de nós)**: "<mqtt_interface_nap>/send/+/+"

- **Transmissão (Gateway envia aos nós)**: "<mqtt_interface_nap>/receive/+/+"

Considerando a recepção, sempre que o Broker recebe uma mensagem de um tópico condizente com a estrutura, a Interface é notificada e recebe a informação. Em seguida, é iniciada a checagem de endereçamento. Se o NID de destino e origem são válidos a requisição é aceita e filtrada para a etapa de roteamento, caso contrário é descartada.

Na transmissão, é feita a checagem de solicitações para transmissões MQTT. Com a condição atingida, a Interface recebe os endereços de transmissor e receptor, montando o tópico para publicação com a mensagem desejada adicionada no pacote. A origem pode ser substituída pelo Gateway ou um nó, mas este último necessita da configuração da rota de repetição para ser permitida.

Os nós precisam conhecer a estrutura dos tópicos para que também sejam capazes de utilizar os recursos do Gateway. Nesta situação o NAP da Interface é importante, já que a sua adição tem o intuito de destacar com qual rede se está comunicando, considerando que o Gateway é o responsável por mantê-la.

3.7 Nós de Rede

Para que o Gateway seja testado, é preciso o uso de nós para estabelecer uma rede de comunicação. Ela foi criada a partir de três kits Wemos Mini D1, cada um atuando individualmente como clientes HTTP, LoRa e MQTT. Estes embarcados são equipados pelo microcontrolador ESP8266 da Espressif Systems (2020), dotados nativamente de diversos recursos, podendo citar portas GPIO, barramentos de comunicação SPI, I2C, UART e Wi-Fi 802.11b/g/n.

A preferência pelo módulo Wemos é em decorrência do Wi-Fi integrado, facilitando o uso do HTTP e MQTT. No entanto, o uso adicional do módulo LoRa se faz necessário. Nesta função foi utilizado o XL1276-P01 atuando como transmissor e receptor (Tx/Rx), que, assim como o 2AD66-LORAV2, utiliza e conserva as mesmas características operacionais do rádio SX1276 da SemTech Corporation. A montagem física e o circuito esquemático de conexão do XL1276-P01 estão representados nas Figuras 46 e 47. Apesar da escolha, qualquer sistema computacional em teoria é capaz de se comunicar com o Gateway, desde que respeite os critérios estabelecidos pelas interfaces e esteja previamente cadastrado.

Todos os três nós foram programados utilizando a Arduino IDE, como sugere Schwartz (2016) com a linguagem de programação C++. O código[1] possui a estrutura básica para receber e transmitir pacotes na rede, podendo ser adaptado conforme necessidade.

[1] Disponível para consulta no repositório: "https://github.com/alexjgUFOP/rasp4iot-nodes"

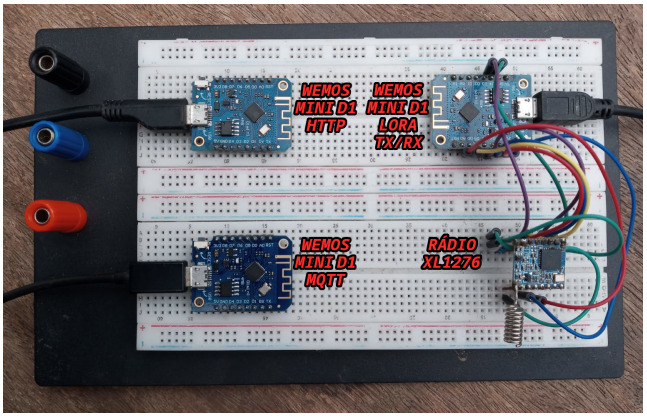

Figura 46 – Montagem dos nós de rede. Fonte: elaboração própria.

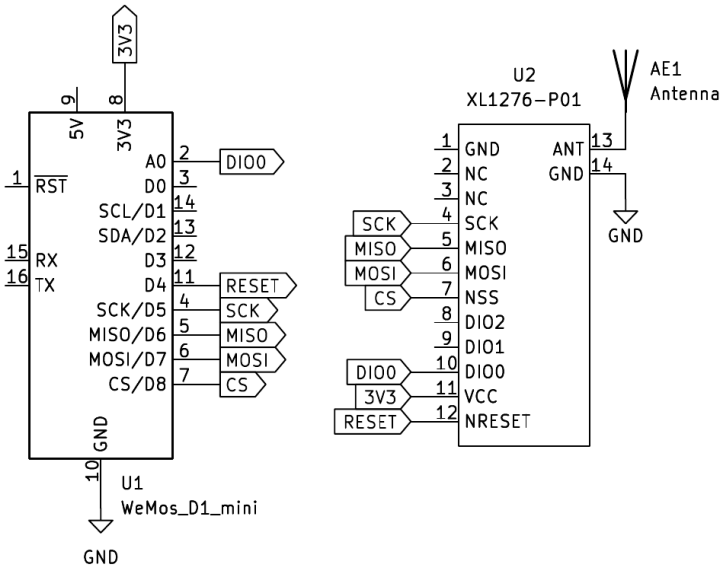

Figura 47 – Circuito esquemático de conexão do LoRa XL1276-P01 no Wemos Mini D1. Fonte: elaboração própria.

3.7.1 Servidor Grafana

Considerando a rede ativa, um volume considerável de pacotes são transportados, o que pode dificultar o acompanhamento do status operacional ao analisar as mensagens em formato de texto. Presumindo este cenário, foi utilizado uma instância do Servidor Grafana externa ao Gateway na avaliação da funcionalidade da API REST, servindo de apresentação gráfica das informações e exemplo de como a API pode ser integrada em demais aplicações.

O Grafana é uma plataforma de código aberto usada para monitoramento gráfico de séries temporais, explica Salituro (2020). Além disso, destaca vantagens de gratuidade e extensões de funções adicionais nos painéis de informações (*dashboards*) via *plugins* elaborados pela comunidade. Os *dashboards* podem ser compostos por um variado conjunto de dados como textos, gráficos, tabelas e números. A Figura 48 contém um típico *dashboard*

Grafana.

Figura 48 – Exemplo de um *dashboard* Grafana. Fonte: Salituro (2020, p. 12).

Após preparação dos módulos ESP8266 Wemos foi criado o *dashboard* "Rasp4IoT Nodes", contendo os gráficos individuais das mensagens armazenadas no banco de dados. Ele é exibido na Figura 49. As requisições HTTP na API são realizadas por meio do *plugin* Infinity 2.4.0 instalado via Grafana. Desta forma, basta o uso do sistema de consulta de mensagens abordado na Seção 3.3.5.

A Figura 50 compartilha o processo de configuração das requisições GET. Os sub-menus "URL", "HTTP Headers" e "URL Query Params" contém as informações obrigatórias para a API. Já o sub-menu "Columns" realiza a conversão de campos presente dos resultados em informação útil. Portanto, os dados (**node_message**) e o instante que o nó realiza o envio (**createdAt**) são transformadas de texto puro aos formatos numéricos e de tempo usados nos gráficos Grafana.

Apesar da requisição anterior utilizar a busca por todos os registros, ela é usada apenas pela simplicidade e análise do desempenho, pois se trata de desperdício de recursos. Esta ação, além de causar sobrecarga desnecessária de memória e processamento do Raspberry Pi, também é composta, na grande maioria, de dados repetidos sobrepostos. Ou seja, é aconselhado o uso dos demais parâmetros fornecidos pela API para buscas mais otimizadas.

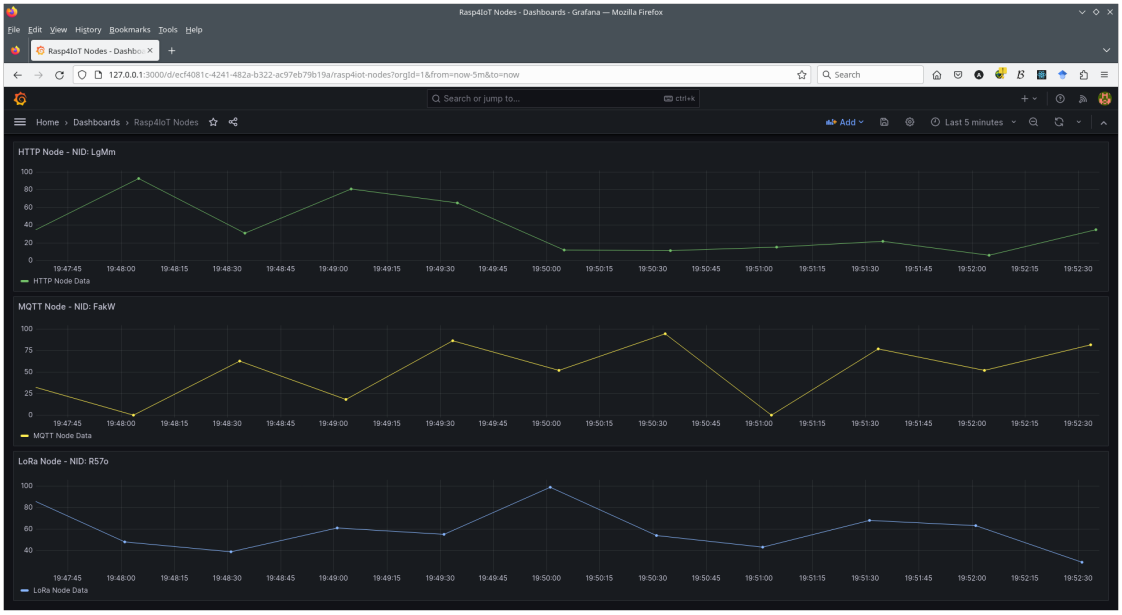

Figura 49 – *Dashboard* Grafana “Rasp4IoT Nodes”. Fonte: elaboração própria.

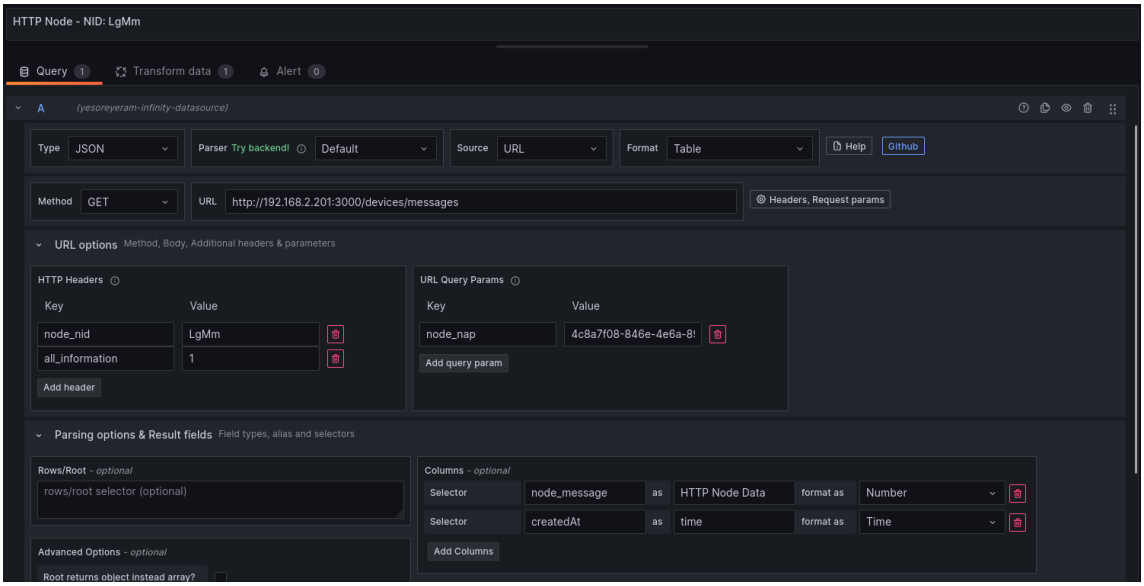

Figura 50 – Requisições GET na API REST via Grafana. Fonte: elaboração própria.

4 Resultados

Ao encerrar o desenvolvimento do projeto, foram coletados os dados referentes aos testes de comunicação do Raspberry Pi atuando como Gateway IoT. Com este intuito, foi utilizado os nós e seus códigos preparados para validar se a plataforma cumpre os objetivos traçados de intercomunicar dispositivos heterogêneos. Pensando nisso, cada Wemos Mini D1 foi configurado a fim de gerar e transmitir números aleatórios (0 a 100) em intervalos de 30 segundos ao Gateway e vizinhos.

Por fim, é feita a checagem da API REST, apurando a disponibilidade dos dados via requisições HTTP no servidor Grafana, checando sua integração com demais aplicações independentes dos usuários.

4.1 Ativação dos nós

Utilizando a gerência Web, os nós de testes foram cadastrados, obtendo em especial os endereços NID e o código NAP, propriedades obrigatórias nas ações de comunicação e manipulação de dados. A Figura 51 mostra o resultado do procedimento entregue pela captura das páginas de interfaces de rede, que dentre as informações presentes, possui a lista de todos os parâmetros dos participantes habilitados.

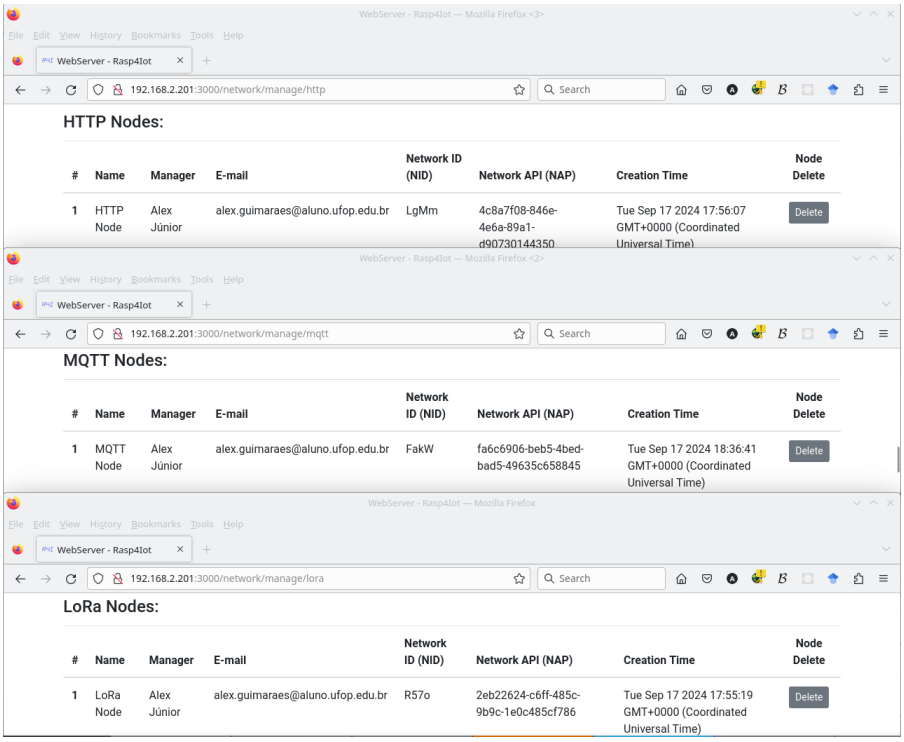

Figura 51 – Cadastro dos nós Wemos Mini D1. Fonte: elaboração própria.

4.2 Configuração dos Wemos Mini D1

A partir das propriedades exibidas nas Figuras 28, 29, 30 e 51, é possível extrair o NID e NAP necessários aos ESP8266 conforme disposto na Tabela 2.

Tabela 2 – Parâmetros de identificação dos nós na rede. Fonte: elaboração própria.

Interfaces de rede		
	NID	NAP
HTTP	wOyo	b36f57ad-0345-4643-be9f-9d4395d4e91c
LoRa	YyeM	4424ee34-daac-4f6d-bcb6-b11a849e4f7a
MQTT	bAS8	16ef0b02-2849-45d5-8e8a-20c5ae3ca5dd
Nós - Wemos Mini D1		
	NID	NAP
HTTP	LgMm	4c8a7f08-846e-4e6a-89a1-d90730144350
LoRa	R57o	2eb22624-c6ff-485c-9b9c-1e0c485cf786
MQTT	FakW	fa6c6906-beb5-4bed-bad5-49635c658845

A inclusão deles no código, ressaltada em destaque na Figura 52, viabiliza que conheçam o próprio endereço e identifiquem quem está apto a se comunicar com eles. Com o ajuste e a posterior gravação nos embarcados, suas integrações com o Gateway são alcançadas.

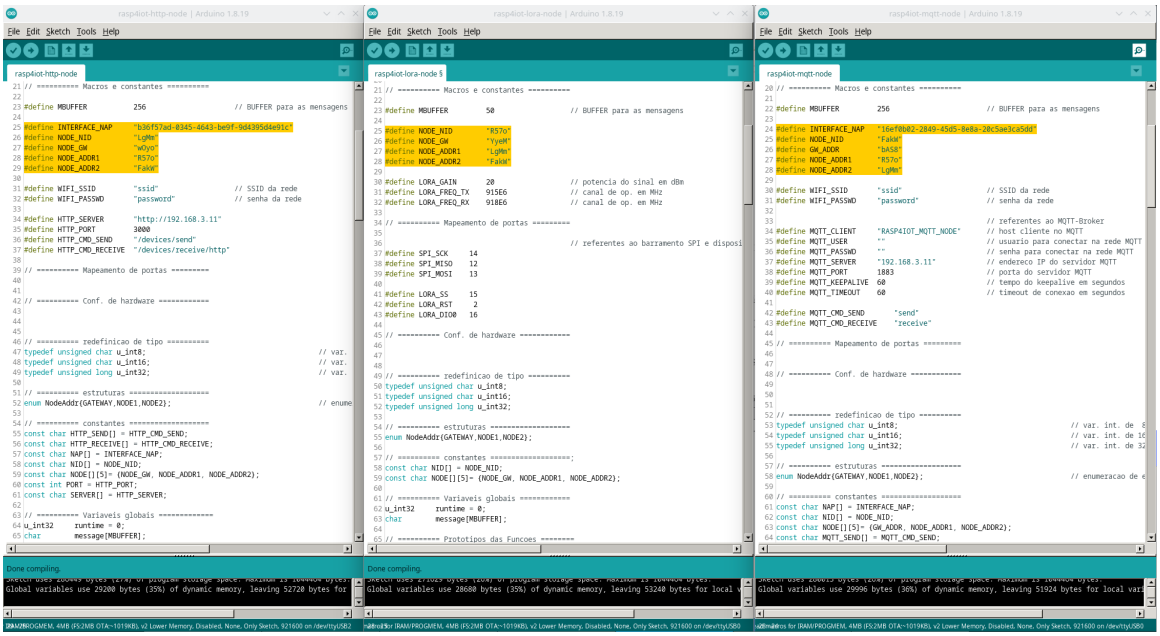

Figura 52 – Configuração de endereçamento. Fonte: elaboração própria.

Neste momento, os nós são aptos a utilizarem os recursos oferecidos no Raspberry Pi, bastando apenas a adição das rotas de repetição para sinalizar ao sistema que eles permitem que demais participantes encaminhem pacotes. A Figura 53 mostra o produto das novas rotas traçadas.

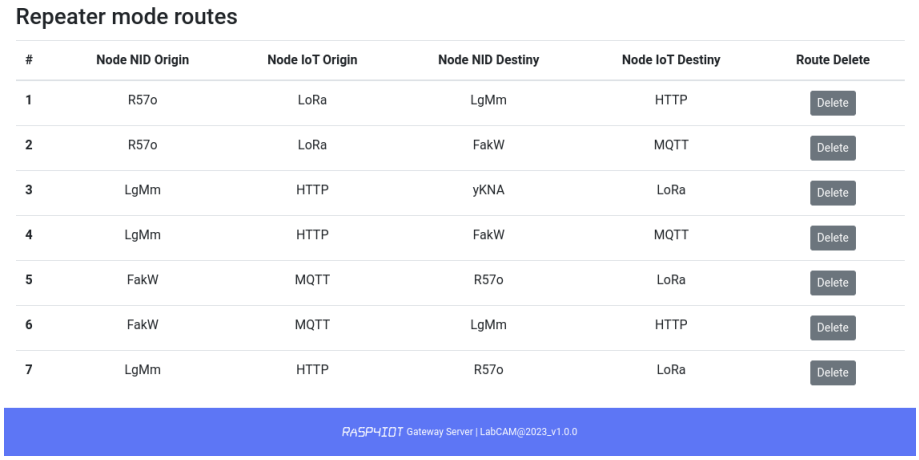

Figura 53 – Rotas de comunicação entre os nós Wemos Mini D1. Fonte: elaboração própria.

4.3 Testes de intercomunicação

Finalizado o procedimento de preparação dos nós, se torna viável o acompanhamento das transmissões em cada microcontrolador envolvido. Checando o registro de cada um pela Arduino IDE via conexão Serial/USB obtêm-se os resultados presentes nas Figuras 54, 55, 56 referentes respectivamente aos módulos Wemos HTTP, LoRa e MQTT.

As cores presentes destacam os endereços NID envolvidos (Tabela 2) e o tipo de operação. Enquanto o amarelo ressalta o envio, as demais ressaltam a recepção. Pode-se observar que todos os vizinhos se comunicaram entre si de maneira transparente, ou seja, se tratam como iguais mesmo não conhecendo o método de transmissão empregado nos outros.

4.4 Consulta de mensagens via API

Por meio do *Dashboard* Grafana "Rasp4IoT Nodes" foi realizado o monitoramento gráfico das transmissões, alcançando a representação demonstrada na Figura 57. Ela contém todas as mensagens enviadas pelos nós HTTP, LoRa e MQTT em um período ininterrupto de aproximadamente sete dias.

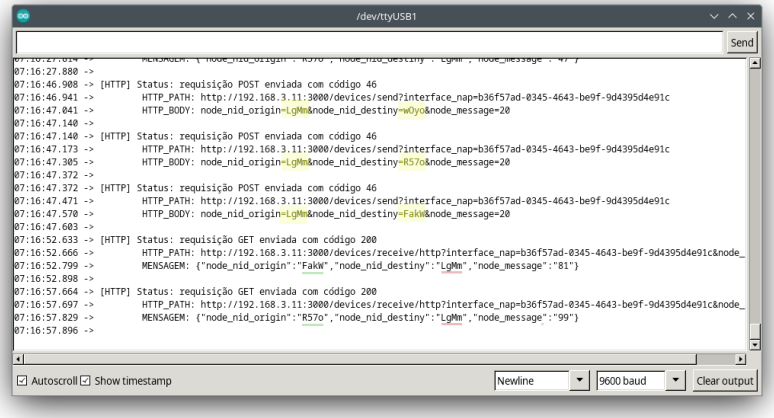

Figura 54 – *Log* de comunicação do Wemos Mini D1 - HTTP. Fonte: elaboração própria.

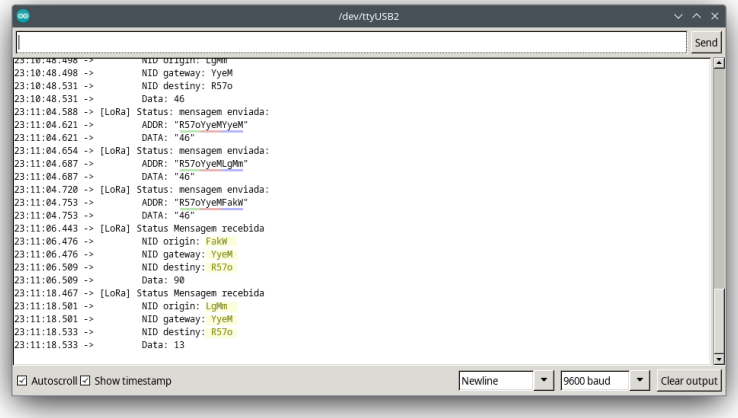

Figura 55 – *Log* de comunicação do Wemos Mini D1 - LoRa. Fonte: elaboração própria.

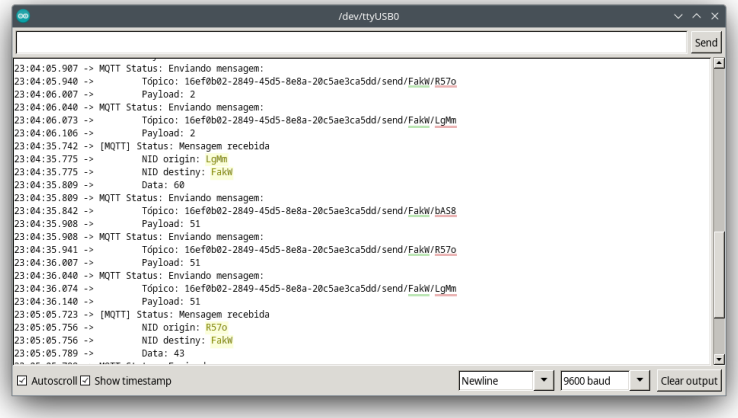

Figura 56 – *Log* de comunicação do Wemos Mini D1 - MQTT. Fonte: elaboração própria.

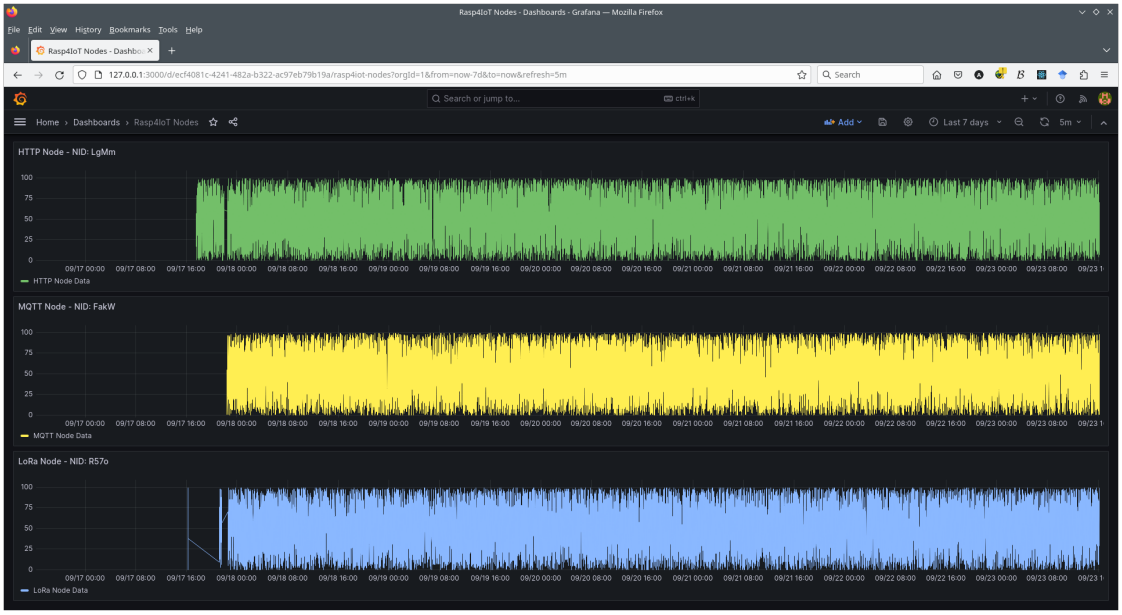

Figura 57 – Gráfico de mensagens dos nós de rede. Fonte: elaboração própria.

4.5 Desempenho do Raspberry Pi

Utilizando o utilitário *"htop"* no Raspbian é possível coletar o resumo do desempenho de *hardware* do Raspberry Pi 3B+ atuando como Gateway. Os resultados exibidos na Figura 58, apresentam o consumo atual de cada um dos quatro núcleos do processador, a memória RAM disponível, ocupando aproximadamente 23% (211 MB de 910 MB), tempo ativo e os processos em execução durante os testes.

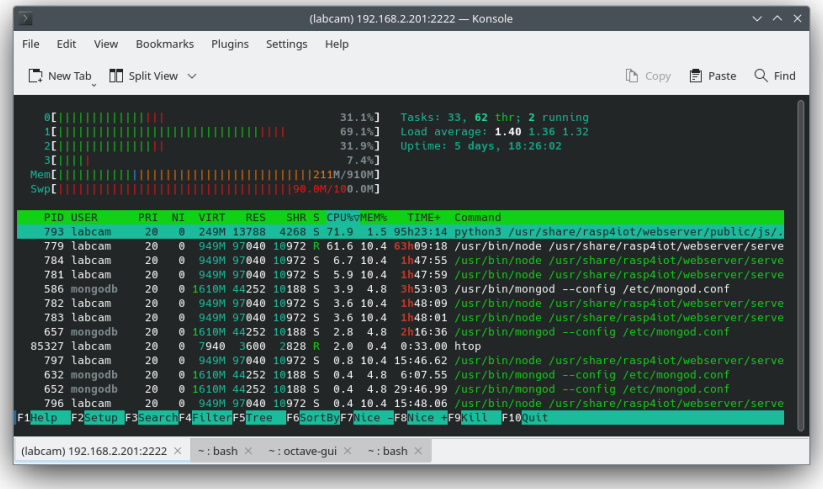

Figura 58 – Desempenho do Raspberry Pi como Gateway. Fonte: elaboração própria.

Em relação ao *load average* (média de carga), foram apresentados os valores 1,40,

1,36 e 1,32. Nemeth et al. (2018) os definem como uma métrica que representa respectivamente a média de quantos processos foram executados nos últimos 1, 5 e 15 minutos. Por consequência, deve ser menor que a quantidade de núcleos, caso contrário, indica que o sistema está sobrecarregado. O *load average* tem a vantagem de ser afetado por demais operações de entrada e saída como o acesso e gravação em disco, ou seja, representa o desempenho do sistema não apenas em questão do uso do processador.

Considerações finais

O intuito principal deste trabalho foi propor um Gateway capaz de gerenciar redes de dispositivos sem fio para aplicações IoT. Para isso, foram investigados meios de tornar prática a configuração e manutenção dessas redes, facilitando a adição de novos equipamentos pelos próprios usuários. Contudo, considerando a grande diversidade de tecnologias, protocolos e meios de comunicação, integrá-los é uma meta desafiadora. Somando ao fato que essas informações precisam ser acessadas na Internet, surge a importância de apresentar métodos de requisitar esses dados com disponibilidade e confiabilidade. Nesse caso, tem-se a necessidade de um sistema independente de outros serviços, uma vez que essa fragilidade pode comprometer a operacionalidade.

Após etapa de pesquisa, a solução foi planejada para uma arquitetura de Gateway IoT construída sobre o Raspberry Pi 3B+ capaz de ser intercomunicável, ou seja, reunir métodos de transmissões heterogêneos. Os resultados mostram que o projeto foi eficaz no cenário com dispositivos HTTP, LoRa e MQTT unidos e trocando informações entre si ao mesmo tempo a partir das interfaces dedicadas. Elas se mostraram eficazes em criar uma camada de abstração das sub-redes com o processamento interno do Gateway, propiciando que novas técnicas sejam incluídas sem comprometimento brusco do modelo alcançado.

O processo de cadastro e configuração de nós necessitou apenas do acesso ao servidor Web. Ele se mostrou útil em consulta rápida de parâmetros e gerenciamento geral. O sistema de login incluído também fornece um meio de proteção contra acessos indevidos.

A partir dos resultados, pode-se deduzir um comportamento sustentável da API REST, atendendo os requisitos desejados. Durante o período de testes não foram percebidas situações em que ela não respondesse as requisições efetuadas, devolvendo como resultado as mensagens que os nós de testes enviaram ao Gateway.

O projeto se apresentou autossuficiente, mantendo todos os elementos fundamentais da arquitetura mesmo sem conexão à Internet. Consequentemente, seu uso é viável para cenários que ele ou os nós só podem ter acesso a rede interna. Contudo existe uma ressalva, a versão atual demanda que o rádio Wi-Fi do Raspberry precise de um ponto de acesso para que os nós se conectem. Embora exista a limitação, com modificações de configurações avançadas de rede no Raspbian este problema pode ser solucionado, dispensando alterações na aplicação.

Não houve tentativas de implementar serviço de criptografia de mensagens para manter a simplicidade dos nós. Porém, se desejado, basta que o *payload* já seja enviado codificado, uma vez que não é necessário conhecer o que se trafega, e sim os endereços envolvidos.

Mesmo rodando serviços como o banco de dados MongoDB, Broker Mosquitto, Python e Node.js, o Gateway obteve bom desempenho. Em todo período de análise o comportamento se manteve estável, com o resultado de *load average*, uso de memória e processamento apoiando a afirmação. Portanto, permite concluir que o projeto se mostra uma alternativa sustentável de método de construção de redes para Internet das Coisas, agilizando e simplificando o processo por já disponibilizar uma estrutura básica a se seguir para conectar dispositivos, se comunicar e ter acesso aos seus dados.

4.6 Sugestões de trabalhos futuros

A respeito de trabalhos futuros, é proposto a elaboração de testes de campo visando obter os limites operacionais máximos do Gateway como quantidade máxima de nós, distância e largura de banda máxima que cada interface é capaz de oferecer. Pensando nisso, deve ser adicionado suporte para visualização de força do sinal (RSSI, Received Signal Strength Indication) e último estado online dos nós na plataforma Web, informações importantes para esse tipo de procedimento.

A interface Web também carece de alguns recursos adicionais para complementar a experiência dos usuários. Como exemplo podem ser citados um sistema de login para multiusuários, *log* de sistema, e melhorias ao suporte de modificação de parâmetros das páginas das interfaces.

Fica sugerido o desenvolvimento de uma *shield* para os módulos LoRa externos, assim como uma *case* de proteção para possibilitar a instalação em ambiente externo. Por fim, deve-se considerar a melhoria do sistema de segurança. A ausência de Firewall pode deixar o Gateway sujeito a ataques de invasão e negação de serviço. Falta de criptografia também pode abrir espaço para falhas de segurança da informação, logo, é razoável a inclusão das versões criptografadas dos protocolos utilizados, tal como o HTTPS.

Referências

ALANI, Mohammed M. OSI Model. In: GUIDE to OSI and TCP/IP Models. Cham: Springer International Publishing, 2014. P. 5–17. ISBN 978-3-319-05152-9. DOI: 10.1007/978-3-319-05152-9_2. Disponível em: https://doi.org/10.1007/978-3-319-05152-9_2. Citado 3 vezes nas páginas 20, 21.

ANATEL. *Anatel - Resolução nº 759, de 19 de janeiro de 2023*. Jan. 2024. [Online; acessado em 25. Jan. 2024]. Disponível em: https://informacoes.anatel.gov.br/legislacao/resolucoes/2023/1834-resolucao-759. Citado 1 vez na página 28.

BANERJI, Sourangsu; CHOWDHURY, Rahul Singha. On IEEE 802.11: wireless LAN technology. *arXiv preprint arXiv:1307.2661*, 2013. Citado 3 vezes nas páginas 25, 26.

BOR, Martin; VIDLER, John Edward; ROEDIG, Utz. LoRa for the Internet of Things. Junction Publishing, 2016. Citado 2 vezes na página 27.

BRADSHAW, Shannon; BRAZIL, Eoin; CHODOROW, Kristina. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, 2019. Citado 3 vezes na página 52.

CHANG, Frank; ONOHARA, Kiyoshi; MIZUOCHI, Takashi. Forward error correction for 100 G transport networks. *IEEE Communications Magazine*, IEEE, v. 48, n. 3, s48–s55, 2010. Citado 1 vez na página 29.

CHEN, Hao; JIA, Xueqin; LI, Heng. A brief introduction to IoT gateway. In: IET. IET international conference on communication technology and application (ICCTA 2011). 2011. P. 610–613. Citado 1 vez na página 16.

COMER, Douglas E. *Redes de computadores e Internet-6*. Bookman Editora, 2016. Citado 2 vezes na página 15.

CROTTI, Y et al. Raspberry pi e experimentação remota. In: INTERNATIONAL Conference on Interactive Computer aided Blended Lerning–ICBL. 2013. Citado 1 vez na página 36.

DEVALAL, Shilpa; KARTHIKEYAN, A. LoRa technology-an overview. In: IEEE. 2018 second international conference on electronics, communication and aerospace technology (ICECA). 2018. P. 284–290. Citado 2 vezes na página 27.

ELKHODR, Mahmoud; SHAHRESTANI, Seyed; CHEUNG, Hon. Emerging wireless technologies in the internet of things: a comparative study. *arXiv preprint arXiv:1611.00861*, 2016. Citado 1 vez nas páginas 15, 24.

ESPRESSIF SYSTEMS. *ESP8266 Technical Reference*. 2020. Accessed: 2024-09-16. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf. Citado 1 vez na página 63.

EVANS, Dave. A internet das coisas. *San José: Cisco IBSG*, 2011. Citado 1 vez na página 18.

FLORES, ADM; RIBEIRO, Luciano Maciel; ECHEVERRIA, Evandro Luiz. A tecnologia da informação e comunicação no ensino superior: Um olhar sobre a prática docente. *Spacios*, v. 38, n. 5, p. 1–14, 2017. Citado 1 vez na página 16.

GARG, Hittu; DAVE, Mayank. Securing iot devices and securelyconnecting the dots using rest api and middleware. In: IEEE. 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU). 2019. P. 1–6. Citado 1 vez na página 37.

GOMES, Ruan D et al. Desafios de redes de sensores sem fio industriais. *Revista de Tecnologia da Informação e Comunicação*, v. 4, n. 1, p. 16–27, 2014. Citado 2 vezes nas páginas 24, 25.

GOURLEY, David; TOTTY, Brian. *HTTP: the definitive guide*. "O'Reilly Media, Inc.", 2002. Citado 5 vezes nas páginas 30–32.

HUNT, Craig. *TCP/IP network administration*. "O'Reilly Media, Inc.", 2002. v. 3. Citado 3 vezes nas páginas 20, 22.

ISLAM, SM Riazul; UDDIN, M Nazim; KWAK, Kyung Sup. The IoT: Exciting possibilities for bettering lives: Special application scenarios. *IEEE Consumer Electronics Magazine*, IEEE, v. 5, n. 2, p. 49–57, 2016. Citado 1 vez na página 15.

KIM, Seong-Min; CHOI, Hoan-Suk; RHEE, Woo-Seop. IoT home gateway for auto-configuration and management of MQTT devices. In: IEEE. 2015 IEEE Conference on Wireless Sensors (ICWiSe). 2015. P. 12–17. Citado 5 vezes nas páginas 38, 39, 44, 54.

LAMPKIN, Valerie et al. *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*. IBM Redbooks, 2012. Citado 2 vezes nas páginas 33, 34.

LEACH, Paul; MEALLING, Michael; SALZ, Rich. *A universally unique identifier (uuid) urn namespace*. 2005. Citado 1 vez na página 44.

LI, Yadong et al. Research based on OSI model. In: IEEE. 2011 IEEE 3rd International Conference on Communication Software and Networks. 2011. P. 554–557. Citado 1 vez na página 20.

LIGHT, Roger A. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, v. 2, n. 13, p. 265, 2017. Citado 1 vez na página 51.

MAKSIMOVIĆ, Mirjana et al. Raspberry Pi as Internet of things hardware: performances and constraints. *design issues*, v. 3, n. 8, p. 1–6, 2014. Citado 1 vez na página 36.

MAURYA, Richa et al. Application of Restful APIs in IOT: A Review. *Int. J. Res. Appl. Sci. Eng. Technol*, v. 9, p. 145–151, 2021. Citado 2 vezes na página 37.

MCCUNE, Earl. Spread spectrum. In: PRACTICAL Digital Wireless Signals. Cambridge University Press, 2010. P. 226–247. (The Cambridge RF and Microwave Engineering Series). Citado 2 vezes na página 26.

MENDES, Daniel. REST no Protheus — Básico do básico - TOTVS Developers - Medium. *Medium*, TOTVS Developers, dez. 2021. Disponível em: https://medium.com/totvsdevelopers/rest-no-protheus-b%5C%C3%5C%A1sico-do-b%5C%C3%5C%A1sico-a4b4431a4e3e. Citado 1 vez na página 37.

MONTANNY, Stéphane. *LoRa-LoRaWAN and Internet of Things*. Université Savoie Mont Blanc, 2022. Disponível em: https://www.univ-smb.fr/lorawan/wp-content/uploads/2022/01/Book-LoRa-LoRaWAN-and-Internet-of-Things.pdf. Citado 3 vezes nas páginas 27–29, 59.

NAIK, Nitin. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: IEEE. 2017 IEEE international systems engineering symposium (ISSE). 2017. P. 1–7. Citado 1 vez na página 33.

NEMETH, Evi et al. UNIX and Linux system administration handbook. *USENIX Open Access Policy*, v. 59, 2018. Citado 1 vez na página 72.

NICERF WIRELESS TECHNOLOGY LTD. *Lora™ V2 User Manual*. 2017. FCC ID: 2AD66-LORAV2. Disponível em: https://fcc.report/FCC-ID/2AD66-LORAV2/3379111. Citado 1 vez nas páginas 56, 57.

ORTIZ, Fernando Molano et al. Caracterizaçao de desempenho de uma rede lora em ambientes urbanos: Simulaçao vs. prática. In: SBC. ANAIS do III Workshop de Computação Urbana. 2019. P. 167–180. Citado 1 vez na página 26.

PARZIALE, Lydia et al. TCP/IP tutorial and technical overview. IBM Redbooks, 2006. Citado 3 vezes nas páginas 22, 23.

PEREIRA, Adriano; OLIVEIRA SIMONETTO, Eugênio de. Indústria 4.0: conceitos e perspectivas para o Brasil. *Revista da Universidade Vale do Rio Verde*, v. 16, n. 1, 2018. Citado 1 vez na página 16.

PICKHOLTZ, Raymond; SCHILLING, Donald; MILSTEIN, Laurence. Theory of spread-spectrum communications-a tutorial. *IEEE transactions on Communications*, IEEE, v. 30, n. 5, p. 855–884, 1982. Citado 1 vez nas páginas 26, 27.

RASPBERRY PI FOUNDATION. *Raspberry Pi 3 B+ Product Brief*. 2023. Disponível em: https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf. Citado 1 vez na página 36.

RUFINO, Nelson Murilo de O. *Segurança em redes sem fio: aprenda a proteger suas informações em ambientes wi-fi e bluetooth*. Novatec Editora, 2019. Citado 1 vez na página 23.

SALITURO, Eric. *Learn Grafana 7.0: A beginner's guide to getting well versed in analytics, interactive dashboards, and monitoring*. Packt Publishing Ltd, 2020. Citado 1 vez nas páginas 64, 65.

SANTOS, Bruno P et al. Internet das coisas: da teoria à prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, v. 31, p. 16, 2016. Citado 3 vezes nas páginas 15, 18, 19.

SCHILLING, Donald L et al. Spread spectrum for commercial communications. *IEEE Communications Magazine*, IEEE, v. 29, n. 4, p. 66–79, 1991. Citado 1 vez na página 26.

SCHWARTZ, Marco. *Internet of Things with ESP8266*. Packt Publishing Ltd, 2016. Citado 1 vez na página 63.

SEMTECH CORPORATION. *LoRa and LoRaWAN: A Tech Overview*. 2019. [Online; acessado em 22. Jan. 2024]. Disponível em: https://lora-developers.semtech.com/uploads/documents/files/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf. Citado 1 vez na página 26.

SEMTECH CORPORATION. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver*. 2020. Disponível em: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE. Citado 5 vezes nas páginas 56–59.

SILVA, Rui. *pyLoRa 0.3.1*. 2019. https://pypi.org/project/pyLoRa/. Accessed: 2024-09-08. Citado 1 vez na página 59.

SOUSA, Marcelo Portela; LOPES, Waslon Terllizzie A. Desafios em redes de sensores sem fio. *dados*, v. 2, p. 3, 2011. Citado 1 vez na página 24.

SYED, Basarat. *Beginning Node. js*. Apress, 2014. Citado 1 vez na página 41.

TELECO. *WLAN X Sistemas Móveis Celulares: Faixas de Frequências*. Jan. 2024. [Online; acessado em 25. Jan. 2024]. Disponível em: https://www.teleco.com.br/tutoriais/tutorialwlanx/pagina_3.asp. Citado 1 vez na página 28.

TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, IEEE, v. 14, n. 6, p. 80–83, 2010. Citado 1 vez na página 41.

VAN KRANENBURG, Rob; BASSI, Alex. IoT challenges. *Communications in Mobile Computing*, Springer, v. 1, n. 1, p. 9, 2012. Citado 2 vezes nas páginas 15, 16.

WONG, Clinton. *Http pocket reference: Hypertext transfer protocol*. "O'Reilly Media, Inc.", 2000. Citado 3 vezes nas páginas 29, 30.

WUKKADADA, Bharati et al. Comparison with HTTP and MQTT in Internet of Things (IoT). In: IEEE. 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). 2018. P. 249–253. Citado 1 vez nas páginas 29, 30.

YANNAKOPOULOS, John. Hypertext transfer protocol: A short course. *University of Crete*, 2003. Citado 1 vez na página 29.

YASSEIN, Muneer Bani et al. Internet of Things: Survey and open issues of MQTT protocol. In: IEEE. 2017 international conference on engineering & MIS (ICEMIS). 2017. P. 1–6. Citado 1 vez nas páginas 32, 33.

YEAGER, Nancy J; MCGRATH, Robert E. *Web server technology*. Morgan Kaufmann, 1996. Citado 1 vez na página 42.