UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL PRANDO

ARQUITETURA IOT DE ALTA DISPONIBILIDADE UTILIZANDO MQTT-SN E RIOT-OS

PATO BRANCO

2023

GABRIEL PRANDO

ARQUITETURA IOT DE ALTA DISPONIBILIDADE UTILIZANDO MQTT-SN E RIOT-OS

High Availability IoT Architecture Using MQTT-SN and RIOT-OS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gustavo Weber Denardin

PATO BRANCO

2023

GABRIEL PRANDO

ARQUITETURA IOT DE ALTA DISPONIBILIDADE UTILIZANDO MQTT-SN E RIOT-OS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de Aprovação: 22 de junho de 2023.

Prof. Dr. Gustavo Weber Denardin
Universidade Tecnológica Federal do Paraná

Prof. Dr. Adriano Serckumecka
Universidade Tecnológica Federal do Paraná

Prof. Dr. Marcelo Teixeira
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2023

Dedico este trabalho aos meus pais e aos meus amigos.

AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão a todos que contribuíram para a realização deste trabalho. Aos meus pais e meu irmão, agradeço por todo o amor e apoio que sempre me deram, sempre foram muito importantes para mim.

Aos meus amigos, Vitor Oliveira, Luan Escudeiro, Cristiano Koxne, Maria Luiza, Wesley de Bona, Gabriel Junges, Daniel Muller e Alfredo Savi, meu profundo agradecimento. Vocês foram fundamentais nessa jornada, seja me ajudando a superar os desafios acadêmicos, seja proporcionando momentos de descontração e amizade que tornaram todo esse processo mais suave.

Um agradecimento especial ao meu orientador, Gustavo Weber Denardin, pela orientação perspicaz, paciência e constante apoio durante esta etapa final do curso. Sua contribuição foi essencial para a realização deste trabalho.

Este trabalho é resultado do apoio que recebi de todos vocês.

“O importante é não parar de questionar. A curiosidade tem razão própria para existir.” (EINSTEIN, 1955)

RESUMO

PRANDO, Gabriel. Arquitetura IoT de alta disponibilidade utilizando MQTT-SN e RIOT-OS. 2023, 63 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2023.

O trabalho aborda o planejamento, implementação, validação e testes de uma arquitetura de comunicação para Internet das Coisas (IoT) que utiliza o protocolo MQTT-SN e o sistema operacional de tempo real RIOT. Propõe-se criar uma rede de sensores em estilo árvore, com múltiplos gateways, utilizando o protocolo de roteamento RPL para definir a melhor rota e o melhor gateway para um dispositivo enviar dados. No desenvolvimento da arquitetura, emprega-se a pilha de protocolos de rede RPL para roteamento e placas com rádio 802.15.4, embora rádios como LoRa, Bluetooth e Zigbee também possam ser utilizados na rede RPL. Aborda-se a integração do MQTT-SN com o RIOT e implementa-se um gateway para traduzir mensagens MQTT-SN para MQTT, permitindo a comunicação com serviços baseados na web. Para tal arquitetura, desenvolve-se um cliente final, que possui uma série de regras para retentativa ou reconexão com gateway, baseado em métricas da rede RPL. Implementa-se também um servidor socket no roteador de borda para a descoberta do endereço do gateway e realizam-se configurações de rede e rotas IPv6. Para testar e validar a arquitetura, utiliza-se o FIT IoT-LAB, uma plataforma que fornece infraestrutura real de dispositivos IoT para testes. Como resultado, observa-se que a arquitetura é eficaz e robusta, os clientes finais conseguem se reconectar a outros gateways em caso de indisponibilidade no gateway que estão conectados, demonstrando resiliência e garantindo alta disponibilidade.

Palavras-chave: IoT; Rede de Sensores; RPL; IEEE 802.15.4; MQTT-SN; RIOT-OS.

ABSTRACT

PRANDO, Gabriel. High Availability IoT Architecture Using MQTT-SN and RIOT-OS. 2023, 63 p. Final Project – Bachelor of Computer Engineering, Federal Technological University of Paraná. Pato Branco, 2023.

The study addresses the planning, implementation, validation and testing of a communication architecture for the Internet of Things (IoT) using the MQTT-SN protocol and the real-time operating system RIOT. It proposes creating a tree sensor network with multiple gateways, utilizing the RPL routing protocol to define the best route and the best gateway for a device to send data. In the development of the architecture, the RPL network protocol stack is used for routing and boards with 802.15.4 radio, though radios like LoRa, Bluetooth, and Zigbee can also be used in the RPL network. It discusses the integration of MQTT-SN with RIOT and implements a gateway to translate MQTT-SN messages to MQTT, enabling communication with web-based services. For this architecture, a final client is developed, which has a set of rules for retrying or reconnecting with the gateway based on RPL network metrics. A socket server is also implemented on the edge router for discovering the gateway address, and network configurations and IPv6 routes are set up. To test and validate the architecture, FIT IoT-LAB is utilized, a platform that provides real IoT device infrastructure for testing. As a result, it is observed that the architecture is effective and robust, and the final clients can reconnect to other gateways in case of unavailability in the gateway they are connected to, demonstrating resilience and ensuring high availability.

Keywords: IoT; Sensors Network; RPL; IEEE 802.15.4; MQTT-SN; RIOT-OS.

LISTA DE CÓDIGOS-FONTE

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
ANSI	American National Standards Institute
API	Interface de Programação de Aplicativos, do Inglês *Application Programming Interface*
CCA	Avaliação de Canal Claro, do Inglês *Clear Channel Assessment*
CLI	Interface de Linha de Comando, do Inglês *Command Line Interface*
DSSS	Espalhamento Espectral por Sequência Direta, do Inglês *Direct Sequence Spread Spectrum*
FTP	Protocolo de Transferência de Arquivos, do Inglês *File Transfer Protocol*
GSM	Sistema Global para Comunicações Móveis, do Inglês *Global System for Mobile Communication*
HTTP	Protocolo de Transferência de Hipertexto, do Inglês *Hypertext Transfer Protocol*
IETF	Internet Engineering Task Force
IP	Protocolo de Internet, do Inglês *Internet Protocol*
IPX	Troca de Pacotes entre Redes, do Inglês *Internetwork Packet Exchange*
LQI	Indicador de Qualidade de Link, do Inglês *Link Quality Indicator*
M2M	Máquina para Máquina, do Inglês *Machine to Machine*
MMU	Unidade de Gerenciamento de Memória, do Inglês *Memory Management Unit*
MPU	Unidade de Proteção de Memória, do Inglês *Memory Protection Unit*
MQTT	Message Queuing Telemetry Transport
MQTT-SN	Message Queuing Telemetry Transport for Sensor Networks
NIB	Base de Informações de Vizinhos, do Inglês *Neighbor Information Base*
OASIS	Organização para a Estrutura de Informação Padrão, do Inglês *Organization for the Advancement of Structured Information Standards*
OSI	Interconexão de Sistemas Abertos, do Inglês *Open Systems Interconnection*
OSPF	Caminho Mais Curto Aberto Primeiro, do Inglês *Open Shortest Path First*
POSIX	Interface Portável de Sistema Operacional, do Inglês *Portable Operating System Interface*
QoS	Qualidade de Serviço, do Inglês *Quality of Service*
RFID	Identificação por Rádio Frequência, do Inglês *Radio-Frequency IDentification*
RIP	Protocolo de Informações de Roteamento, do Inglês *Routing Information Protocol*
RPL	Protocolo de Roteamento para Redes de Baixo Poder e Perda, do Inglês *Routing Protocol for Low-Power and Lossy Networks*
RSMB	Broker de Mensagens Realmente Simples, do Inglês *Really Simple Message Broker*
RSSI	Indicador de Força do Sinal Recebido, do Inglês *Received Signal Strength Indicator*
SDK	Kit de Desenvolvimento de Software, do Inglês *Software Development Kit*
SNMP	Protocolo Simples de Gerenciamento de Rede, do Inglês *Simple Network Management Protocol*
SPX	Sequenced Packet Exchange
SSH	Shell Seguro, do Inglês *Secure Shell*
SSL	Camada de Soquetes Seguros, do Inglês *Secure Sockets Layer*
TLS	Segurança da Camada de Transporte, do Inglês *Transport Layer Security*
UDP	User Datagram Protocol
UWB	Ultra Wide Band

WSN	Rede de Sensores Sem Fio, do Inglês *Wireless Sensor Network*
ZB	Zettabytes

SUMÁRIO

1 INTRODUÇÃO

A área de (Internet das Coisas) tem apresentado um crescimento significativo na última década, devido, principalmente, a expansão da Internet e de aplicações distribuídas. Os dispositivos IoT utilizam, em sua maioria, a comunicação sem fio, entre os protocolos mais aplicados encontram-se o IEEE 802.11, Bluetooth, ZigBee, GSM (Global System for Mobile Communication), RFID e LoRaWAN (ASPENCORE; 2019). Em paralelo a este desenvolvimento, cada vez mais dispositivos inteligentes estão sendo lançados no mercado, na maioria das vezes, conectados à internet e com algum meio de telemetria ou comunicação. Devido a essa difusão, estima-se que o número de dispositivos conectados atinja a marca de 50 bilhões até meados de 2023 e continue a aumentar progressivamente nos próximos anos, ultrapassando 75 bilhões de dispositivos conectados (VAILSHERY, 2016).

Este contexto de crescimento da Internet e de dispositivos interconectados ocasionou um grande aumento na geração de dados. Estima-se que até 2025 sejam gerados mais de 163 zettabytes (ZB) de dados no mundo (IDC; 2021). Em aplicações IoT, por exemplo, é comum a troca de mensagens entre diferentes dispositivos, o que é crucial para que um dispositivo tenha conhecimento do estado do outro e, assim, possa efetuar uma ação de controle. Os protocolos de comunicação sem fio mencionados anteriormente, são vitais para a operação dos dispositivos IoT, pois fornecem a infraestrutura necessária para a transferência de dados. No entanto, quando se trata de facilitar a troca de mensagens entre dispositivos IoT de maneira eficiente, o protocolo MQTT tem se mostrado uma escolha popular. O MQTT, que funciona sobre esses protocolos de comunicação sem fio, é particularmente adequado para aplicações IoT devido à sua leveza e bom desempenho mesmo em conexões com baixa capacidade de tráfego de dados.

O MQTT-SN (MQTT for Sensor Networks) foi desenvolvido como uma extensão do MQTT, como uma alternativa para atender de maneira mais eficiente às necessidades específicas dos dispositivos IoT que utilizam redes de sensores sem fio. Esse protocolo é otimizado para comunicações em ambientes com recursos limitados, como dispositivos de baixo consumo energético e redes com latência variável. Diferentemente do MQTT, que foi projetado para redes TCP/IP, o MQTT-SN foi desenvolvido para ser compatível com redes não IP, como ZigBee, IEEE 802.15.4 e outras redes de sensores. Essa característica permite que o MQTT-SN seja uma solução eficiente e eficaz para dispositivos IoT com diferentes tipos de comunicação e requisitos de rede.

Entretanto para que um projeto de IoT apresente sucesso, é necessário escolher ade-

quadamente os protocolos de comunicação a fim de minimizar os custos. Um fator importante nesse aspecto, e um dos principais contribuintes para o custo da aplicação, está relacionado à quantidade de dados transferidos. Esse custo pode ser parcialmente mitigado pela escolha de uma maneira eficiente de comunicação entre dispositivos e módulos (U-BLOX; 2020).

1.1 OBJETIVO GERAL

Desenvolver uma arquitetura genérica para aplicações IoT resiliente e tolerante a falhas, com clientes implementando protocolo MQTT-SN.

1.2 OBJETIVOS ESPECÍFICOS

1. Projetar uma arquitetura de IoT resiliente e tolerante a falhas;
2. Implantar uma rede de sensores para validar a arquitetura;
3. Balancear carga de clientes entre *gateways* quando houver indisponibilidade em um *gateway*;
4. Enviar dados do *gateway* para nuvem;

1.3 ESTRUTURA DO TRABALHO

Esse trabalho inicia com uma breve abordagem sobre o cenário de aplicações IoT na seção 2.1. Na sequência, os demais tópicos do capítulo 2 tratam sobre conceitos que envolvem uma aplicação IoT, passando por arquitetura, protocolos de comunicação, sistemas operacionais específicos para IoT e nuvem. Já na seção 3 é descrito a metodologia e materiais utilizados no trabalho. Por fim, na seção 4 e 5 encontra-se os resultados e conclusão.

2 REFERENCIAL TEÓRICO

O termo Internet das coisas, mais conhecido como IoT, vem se popularizando cada vez mais nos últimos anos. Foi proposto pelo britânico Kevin Ashton em 1999, que utilizou o nome como título de uma apresentação a fim de chamar a atenção de executivos da empresa ProcterGamble (ASHTON, 2010). IoT define uma rede de dispositivos físicos, *softwares* e outras tecnologias interconectados para comunicação entre si e com a Internet. Sua função visa automatizar, monitorar e controlar aplicações, produtos e locais, sendo que poderá ser utilizado em várias áreas, como: eletrodomésticos, câmeras, etc (ATZORI; IERA; MORABITO, 2010).

O crescimento do uso de dispositivos que utilizam Internet das coisas tem gerado grande impacto social e econômico no planeta. Em uma análise do instituto global McKinsey, de 2015, analisando mais de 150 aplicativos específicos de IoT existentes que poderiam estar em uso generalizado em 10 anos, estimaram que eles poderiam ter um impacto econômico total de 3,9 a 11,1 trilhões de dólares por ano em 2025. No entanto, o crescimento do mercado de IoT acabou sendo afetado pela disseminação do vírus SARS-CoV-2, causador da COVID-19, que resultou em uma pandemia global. Olhando para o futuro, os especialistas preveem que até 2030 o cenário fabril será responsável pela maior parte do valor econômico gerado pela IoT. Estima-se que ele possa gerar entre US1,4$trilhoeUS$ 3,3 trilhões, representando cerca de 26% do total. O principal potencial nesse cenário está na otimização das operações de manufatura, gestão de ativos e gestão de pessoas (MANYIKA, 2015).

Uma das aplicações de IoT mais importantes são as cidades inteligentes. Para aumentar a eficiência, utilizam-se tecnologias de informação e comunicação, para trocar dados e para melhorar a qualidade dos serviços e o bem-estar dos usuários (PRADHAN, 2010). Segundo Hammi *et al.* (2018), uma cidade inteligente é definida por arquitetura que interconecta estruturas físicas, tecnológicas, sociais e de negócios, tudo com objetivo de impulsionar o desenvolvimento coletivo. Também pode ser caracterizada por uma grande implantação de IoT, principalmente por aplicações M2M (*machine to machine*), que interconectam toda sua infraestrutura, formando um ecossistema inteligente e acessível por meio de diversas plataformas comuns aos habitantes. Um exemplo de aplicação a cidades inteligentes é o programa de Rede Elétrica Inteligente, desenvolvido pela Copel no estado do Paraná, cujo objetivo é de monitorar a rede de energia elétrica, facilitando ao cliente o acompanhamento do seu consumo de energia por um aplicativo de celular e de quedas de energias nas localidades, quando houver, o medidor inteligente avisa a Copel imediatamente, que pode agir e encaminhar equipes para verificar a situação, tendo assim

períodos menores de indisponibilidade aos seus clientes (COPEL; 2020).

Com sensores e dispositivos de IoT coletando informações em tempo real ou próximo a isso, um grande volume de dados é gerado. Normalmente, esses dados são armazenados em algum local para serem posteriormente analisados. Devido ao grande volume, muitas vezes complexos, tem-se uma problemática para processar esses dados.

2.1 BIG DATA

Além do crescimento no volume de dispositivos e aplicações IoT, teve-se uma crescente na geração de dados, pois quanto mais aplicações, mais dados de telemetria, monitoramento, etc, são gerados. Um conjunto de dados heterogêneos estruturados, ou não, que não conseguem ser processados por plataformas de processamento de dados é chamado de *Big Data*. Pode ter vários terabytes, petabytes ou até zetabytes. A quantidade de dados que define o *big data* hoje pode nunca atingir seus limites no futuro, pois as tecnologias de captura e armazenamento estão evoluindo. Além disso, um determinado volume de dados não estruturados pode ser denominado como *Big Data*, enquanto um mesmo volume de dados estruturados pode não ser definido como *Big Data*. Isso significa que a definição de *big data* vai além do domínio do volume de dados, depende também de características, como diversidade e velocidade (KAUR; SOOD, 2017).

Não basta apenas armazenar dados de forma estruturada ou não. Manter esses dados inativos é improdutivo. O valor é gerado a partir dos dados no momento em que são analisados e explorados em benefício de um negócio, desbloqueando um grande potencial para a tomada de decisões. Isso transforma um grande volume de dados em fatos e hipóteses concretos (GANDOMI; HAIDER, 2015).

À medida que os dados gerados pelos dispositivos IoT aumentam, a escolha do protocolo de comunicação adequado se torna ainda mais crucial. Na próxima seção, serão discutidos alguns dos principais protocolos de comunicação usados em IoT e como eles são fundamentais para lidar com os desafios apresentados pelo Big Data.

2.2 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação são um conjunto de regras que definem como as aplicações IoT implementam e padronizam a troca de informações entre si.

O protocolo determina como as mensagens trafegarão na rede, qual formato e método

de propagação. Os elementos que compõem são a sintaxe, que define o formato e níveis de sinal; a semântica, que trata das informações de controle para o arranjo e tratativa de erros; por fim a temporização, que está relacionada a velocidade de seguimento dos dados (SILVA JUNIOR, 2021).

Com o surgimento dos protocolos de comunicação, surgiram alguns modelos de referência para padronizar e garantir a comunicação entre sistemas. Dentre eles, um modelo de referência consolidado é o de camadas OSI, que se baseia em uma pilha de protocolos independentes (TANENBAUM, 2003).

2.2.1 Modelo OSI de camadas

O modelo OSI é uma convenção para protocolos de rede, que visa definir as etapas e tarefas necessárias para conectar dois ou mais dispositivos. É definido por 7 camadas, ilustrado na Figura 1, em que cada camada pode ter várias subcamadas (LI *et al.*, 2011). Sendo definidas como:

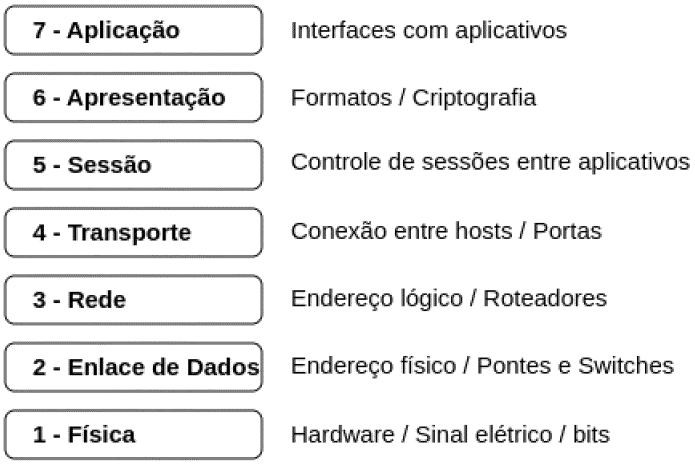
- **Física** - A camada física define especificações elétricas e físicas dos dispositivos. Em especial, define a relação entre um dispositivo e um meio de transmissão, tal como um cabo de cobre ou um cabo de fibra óptica. Isso inclui o *layout* de pinos, tensões, impedância da linha, especificações do cabo, temporização, repetidores, adaptadores de rede e muito mais. É responsável por definir se a transmissão pode ser ou não realizada nos dois sentidos simultaneamente. Diz respeito a transmissão e recepção do fluxo de bits brutos não-estruturados em um meio físico. O fluxo de bits pode ser agrupado em palavras de código ou símbolos e convertido em um sinal físico que é transmitido através de um meio de transmissão de hardware (TANENBAUM, 2003).
- **Enlace de Dados** - A camada de enlace de dados usa os serviços da camada física abaixo dela para enviar e receber bits por canais de comunicação (possivelmente não confiáveis) que podem perder dados. Ele tem uma série de funções, incluindo:
  - Fornecimento de uma interface de serviço bem definida para a camada de rede;
  - Enquadramento de sequências de bits/bytes;
  - Detecção e correção de erros de transmissão;
  - Regular o fluxo de dados para que os receptores lentos não sejam sobrecarregados por remetentes rápidos, entre outras funções.
- **Rede** - está relacionada a transmissão, nível de congestionamento da rede, qualidade

de serviço e roteamento para determinar o custo e melhor caminho para trafegar de um ponto para outro. Quando um dado é enviado de uma origem a um destino, conforme a saturação da rede as mensagens podem ter uma priorização ou rota diferente, a camada de rede é responsável por gerenciar e controlar o roteamento entre a origem e destino do pacote. Dentre os protocolos da camada de rede temos: IP, IPX, RIP, OSPF, etc.

- **Transporte** - responsável por fazer o transporte de dados entre máquinas, é independente de aplicação e rede físicas. Reúne protocolos de transporte M2M, dentre eles os mais utilizado são: TCP/IP, com garantia de entrega de mensagens; UDP mais rápido que TCP, porém sem garantia de entrega de mensagens. Para ocorrer o transporte dos dados, é essencial que as máquinas consigam se comunicar ente si. Protocolos da camada de transporte incluem: TCP, UDP, SPX.
- **Sessão** - se para o transporte de informações M2M, requer que as maquinas se comuniquem entre si, a camada de sessão é responsável por isso. Nesta etapa, pode ser feita verificação nos pontos para obter a sincronização de dados e estabelecer links de comunicação durante a sessão, está entre as funções dessa camada. Basicamente a camada de sessão permite que diferentes máquinas estabeleçam sessões entre usuários.
- **Apresentação:** tradutor entre a rede de dados e o programa. Nesta etapa, caso necessário, há a conversão, compressão e criptografia. Define uma série de códigos para garantir que a fonte de dados possa ser identificada e interpretada no destino.
- **Aplicação** - nível mais alto das aplicações de software, tem como principal função, fornecer interface para que os programas acessem serviços de rede. A camada de aplicação fornece serviços que incluem transferência de arquivos, gerenciamento de arquivos e processamento de informações por e-mail. Entre os protocolos presentes nessa camada estão: Telnet, FTP, HTTP, SNMP, MQTT.

No contexto de IoT, alguns protocolos que seguem o modelo OSI são particularmente relevantes, como o MQTT. O MQTT é um protocolo de camada de aplicação, operando na camada mais alta do Modelo OSI. Ele foi projetado para ser leve e minimizar o uso da rede, o que é crucial para aplicações IoT em que a largura de banda e a eficiência da bateria são considerações importantes.

Figura 1 – Camadas do Modelo OSI



Fonte: adapdato de TANENBAUM (2003).

2.3 MQTT

O protocolo MQTT, criado em 1999 por Andy Stanford-Clack da IBM e Arlen Nipper da Arcom, e somente em 2013 foi padronizado no OASIS (LOCKE, 2010). É um protocolo de transporte de mensagens de publicação/assinatura, que tem um ou mais elementos que publicam em um tópico e um ou mais elementos de interesse que se inscrevem para receber todas as publicações. É leve, simples e projetado para ser de fácil utilização. Essas características o tornam ideal para uso em muitas situações, como em aplicações restritas, com baixo poder de processamento e largura de banda limitada, sendo realidade na grande maioria das aplicações IoT (GUPTA; 2018).

2.3.1 Arquitetura do protocolo

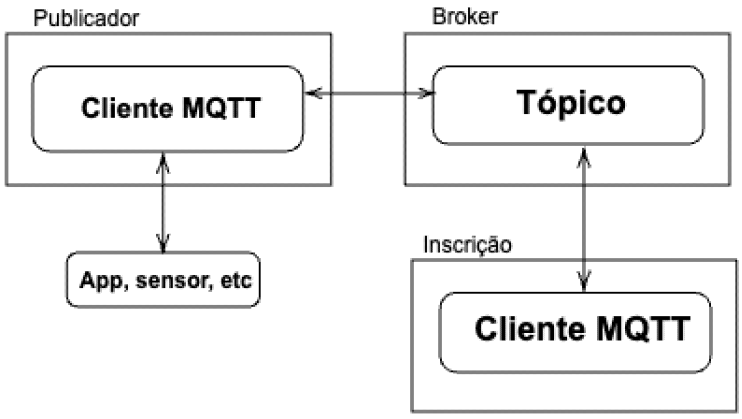
O padrão de publicação e assinatura do protocolo MQTT, também é conhecido como Pub/Sub, ilustrado na Figura 2. Nessa arquitetura é possível observar 3 componentes principais: Publicador (Publisher), Broker (Gerenciador intermediário) e Subscriber (Inscrição). Segundo Xu, Mahendran e Radhakrishnan (2016), cada componente é definido por:

- **Publisher** – dispositivos que geram dados, que são publicados seguindo o formato de tópicos;
- **Broker** – O intermediário age como um nó central realizando a comunicação entre

publicadores e assinantes;

- **Subscriber** – Os assinantes recebem as mensagens publicadas, conforme o tópico em que se inscreveram.

Figura 2 – Arquitetura MQTT



Fonte: Autoria própria (2023).

Um publicador abre conexão com o *broker* e específica um tópico para fazer publicação de mensagens, por outro lado, temos um ou mais *subscriber* que abrem conexão com o *broker* e se inscrevem em tópicos de interesse, a partir do momento que um *subscriber* assina um tópico, irá receber todas as mensagens publicadas no mesmo. Os tipos de mensagens disponíveis tanto para publicador quanto para assinante são apresentadas na Tabela 1.

Tabela 1 – Mensagens protocolo MQTT

Valor	Nome	Direção	Descrição
0	Reservado	Proibido	Reservado
1	CONNECT	Cliente para Servidor	Requisição de cliente
2	CONNACK	Servidor para o Cliente	Reconhecimento de conexão
3	PUBLISH	Cliente para o Servidor	
4	PUBACK	Cliente para o Servidor	
5	PUBREC	Publicação recebida	Publicação recebida
6	PUBREL	Cliente para Servidor	
7	PUBCOMP	Cliente para Servidor	
8	SUBSCRIBE	Cliente para Servidor	Pedido de inscrição
9	SUBACK	Servidor para cliente	Reconhcimento de inscrição
10	UNSUBSCRIBE	Cliente para Servidor	Pedido de desinsção
11	UNSUBACK	Servidor para cliente	Reconhecimento desinscrição
12	PINGREQ	Requisição	Requisição PING
13	PINGRESP	Servidor para cliente	Resposta PING
14	DISCONNECT	Cliente para Servidor	Cliente esta desconectado
15	Reservado	Proibido	Reservado

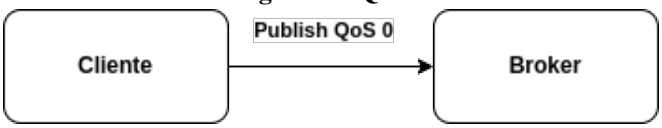
Fonte: Autoria própria (2023).

A conexão entre cliente e *broker* é criado utilizando protocolo TCP, tendo a opção de autenticação por meio de usuário e senha, e com criptografia (SSL/TLS). Também é possível estabelecer conexão de outras formas, como, por exemplo, MQTT rodando sob links seriais (BARROS, 2015).

Cada processo de conexão também estabelece um nível de qualidade de serviço desejado (QoS, Quality of Service), apontando como deve ser a relação entre os elementos que se comunicam. Existem três níveis de qualidade de serviço:

- **QoS 0 (*No máximo uma vez*)** - É semelhante ao protocolo de transporte UDP, que não possui uma confirmação de mensagem, o cliente somente envia a mensagem ao *broker*, conforme ilustrado na Figura 3. Os remetentes não precisam armazenar mensagens para retransmissões futuras.

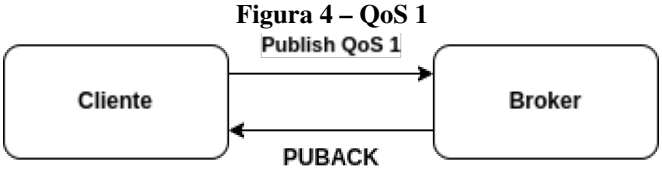
Figura 3 – QoS 0



Fonte: adapdato de HIVEMQ (2015).

- **QoS 1 (*Pelo menos uma vez*)** - Há a confirmação de entrega de uma mensagem, devido o remetente enviar mensagens análogas, ilustrado na Figura 4. Com isso pode ocorrer um atraso no recebimento de feedback, neste caso, é garantido, que uma das mensagens terá o reconhecido bem-sucedido. Para isso a mensagem é armazenada pelo remetente até confirmação de recebimento;

Figura 4 – QoS 1

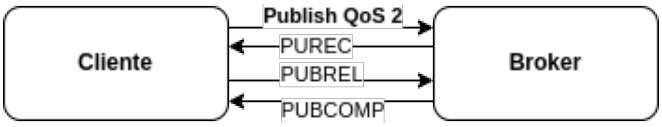


Fonte: adapdato de HIVEMQ (2015).

- **QoS 2 (*Exatamente uma vez*)** - Garante que a mensagem só será entregue uma única vez, com envio de confirmações de recebimento e confirmações de recebimento de confirmações de recebimento, se tem uma confirmação bidirecional conforme ilustra a Figura 5. Similar a confirmação do protocolo de TCP.

Não existe um QoS melhor ou pior, tudo depende do caso de uso da aplicação, recursos disponíveis, etc. O nível de QoS é acordado entre *client* e *broker*, logo em um mesmo tópico podemos ter níveis de QoS heterogênios. Embora o broker represente um ponto de falha na rede

Figura 5 – QoS 2



Fonte: adapdato de HIVEMQ (2015).

ao centralizar as comunicações, ele permite o desacoplamento entre as partes comunicantes, o que não é possível nos modelos de comunicação cliente/servidor. Uma saída para mitigar esse ponto de falha é usar *brokers* clusterizados com várias instâncias rodando em diferentes máquinas e regiões (BARROS, 2015).

Usando um protocolo como o MQTT, é possível interconectar dispositivos e gerenciar o tráfego de dados entre eles. Um exemplo de aplicação que explora esse contexto, e que é relevante para este trabalho, são as redes de sensores.

2.4 REDES DE SENSORES

Uma rede de sensores consiste em um grande número de dispositivos sensores ou atuadores operados por bateria, sendo geralmente equipados com uma quantidade limitada de recursos de armazenamento e processamento. As Redes de Sensores Sem Fio (WSN) podem servir a diferentes propósitos, desde medição e detecção até automação e controle de processos (STANFORD-CLARK; TRUONG, 2013).

2.4.1 MQTT-SN

O protocolo MQTT é otimizado para comunicação em redes que a largura de banda é limitada. O ponto negativo é que por operar sob o protocolo TCP/IP, necessita de conexão ordenada sem perdas, sendo complexo para dispositivos simples e limitados. O MQTT-SN (MQTT for Sensor Networks) é uma variante do MQTT desenvolvida especificamente para redes de sensores sem fio, que geralmente têm restrições de recursos, como limitações de energia, largura de banda e capacidade de processamento. O MQTT-SN foi projetado para ser mais leve e adequado para essas condições do que o MQTT padrão. Existem várias diferenças entre o MQTT e o MQTT-SN. Aqui estão algumas das principais:

1. **Tópicos de tamanho fixo**: MQTT-SN permite o uso de IDs de tópicos de dois bytes em vez de strings de tópicos completas para reduzir a quantidade de dados

transmitidos.

2. **Adaptação para redes sem fio**: MQTT-SN inclui mecanismos para lidar com o adormecimento do dispositivo e a perda de conexão, o que é comum em redes sem fio.
3. **Suporte para descoberta de gateway**: O MQTT-SN inclui mecanismos para permitir que os dispositivos descubram gateways MQTT-SN disponíveis, algo que não está presente no MQTT padrão.

Dessa forma, o MQTT-SN pode ser considerado uma versão do MQTT adaptada às peculiaridades de um ambiente de comunicação sem fio. No entanto, apesar dessas diferenças, o MQTT-SN mantém a maioria dos princípios fundamentais do MQTT, incluindo o modelo de publicação/assinatura e o uso de Quality of Service (QoS) para garantir a entrega de mensagens. . Originalmente foi feito para funcionar com ZigBee, mas pode operar sob qualquer protocolo de rede que tenha um serviço de transferência de dados bidirecional entre um cliente e um gateway (STANFORD-CLARK; TRUONG, 2013).

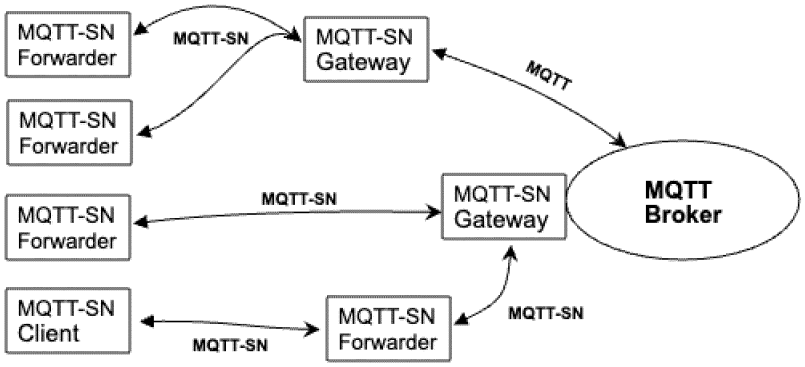
Na Figura 6, observa-se que a arquitetura do MQTT-SN existem 3 componentes principais, que Stanford-Clark e Truong (2013) definem como:

- **Clients** - Dispositivos finais da rede de sensores que podem publicar ou receber dados dos gateways MQTT-SN ou *Forwarder* por meio do protocolo MQTT-SN;
- **Gateways** - Faz a tradução de dados entre clientes MQTT-SN ou *Forwarder* para *brokers* MQTT;
- **Forwarder** - Caso o *Gateway* não esteja na mesma rede do cliente, pode-se utilizar um *forwarder* para encaminhar e receber dados entre cliente e *gateway*. Encapsula os quadros MQTT-SN que recebe do cliente e os encaminha sem alteração para o *gateway*. Também recebe dados do *getway* encapsulado, desencapsula os quadros e os envia aos clientes sem alteração.

Conforme os dados são traduzidos pelo gateway entre MQTT e MQTT-SN, como ilustrado pela Figura 7, pode-se classificar os gateways em dois tipos (STANFORD-CLARK; TRUONG, 2013):

- **Gateway transparente** - Tem sua implementação mais simples, pois, mantém conexão de cada cliente MQTT-SN com o *broker* MQTT. A troca de mensagens é mantida o mais próximo ao original, sendo que a única mudança feita pelo gateway é a tradução de sintaxe entre os dois protocolos. Todas as funções e funcionalidades implementadas pelo servidor podem ser oferecidas ao cliente;

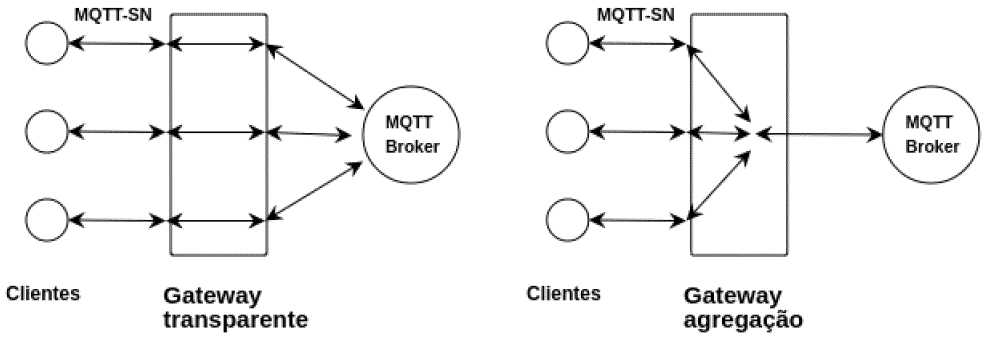
Figura 6 – Arquitetura MQTT-SN



Fonte: adaptado de Stanford-Clark e Truong (2013).

- **Gateway de agregação** - Mantém somente uma conexão MQTT com o *broker* e decide quais informações são fornecidas. Tem implementação mais complexa, porém, pode manter um número maior de dispositivos conectados em uma rede de sensores, reduzindo o número de conexões com o *broker*.

Figura 7 – Gateway transparente e de agregação



Fonte: adaptado de Stanford-Clark e Truong (2013).

Para a implementação do protocolo MQTT-SN, como mencionado anteriormente, é necessário o suporte de outros protocolos de comunicação, especialmente na camada de transporte. No contexto das redes de sensores, dois protocolos relevantes são o 6LoWPAN e o RPL (*IPv6 Routing Protocol for Low Power and Lossy Networks*), que desempenham papéis fundamentais na comunicação eficiente e confiável em ambientes de IoT. Além disso, o padrão IEEE 802.15.4 é frequentemente utilizado como a base para a camada física e de link de dados nessas redes, devido ao seu baixo consumo de energia, suporte para baixas taxas de dados e mecanismos de

confiabilidade. Outro ponto é sua flexibilidade em topologias de rede e padronização que facilita a interoperabilidade entre dispositivos.

2.5 IEE 802.15.4

O protocolo IEEE 802.15.4 define a estrutura da camada física e as subcamadas MAC e LLC em redes LowPan, que incluem dois tipos de dispositivos: Full Function Devices (FFDs) e Reduced Function Devices (RFDs). FFDs são mais complexos e tem maior consumo de energia, mas têm a capacidade de se comunicar com ambos os tipos de dispositivos. RFDs, em contraste, têm menor complexidade e consumo de energia, mas só podem se comunicar com FFDs.

Dentro dessa estrutura, há três tipos de dispositivos lógicos: coordenador, roteador e terminal. O Coordenador da Rede é um dispositivo crucial que cria e mantém a rede, gerenciando elementos importantes, como as tabelas de roteamento. Em uma rede, só deve existir um Coordenador ativo de cada vez. Além disso, os Coordenadores têm a capacidade de se comunicar com redes externas, servindo como gateways para transferir dados entre redes diferentes. O Roteador, que geralmente é um FFD, desempenha um papel intermediário, encaminhando mensagens entre o Coordenador e os dispositivos Terminais. Estes últimos são os dispositivos mais básicos conectados à rede, geralmente utilizados para monitorar variáveis ambientais. Os Terminais são geralmente RFDs e tendem a estar presentes em maior número numa rede (JAHNO, 2016).

Devido ao alcance desses dispositivos não ultrapassar uma faixa de cem metros, em uma rede sem interferências, o uso de dispositivos roteadores é essencial para aumentar a área de cobertura de uma rede de sensores.

2.5.1 Camada Física

A camada física do IEEE 802.15.4 é capaz de operar em várias bandas de frequência. Sendo a mais comum a de 2,4 GHz, por ser globalmente aceita. Nessa banda, o IEEE 802.15.4 define 16 canais, cada um com uma largura de 5 MHz, tendo taxas de transmissão de 250 kbps. Para a camada física o IEEE 802.15.4 utiliza modulação DSSS para garantir transmissões robustas, e possui recursos como RSSI, LQI e CCA para monitorar a qualidade do sinal e disponibilidade do canal. Composta por cabeçalho e carga útil, os quadros podem ter até 127 bytes de dados. Essa camada é otimizada para baixo consumo de energia e taxas de dados moderadas,

adequando-se perfeitamente a aplicações de IoT e redes de sensores.

2.5.2 Camada de Enlace

A camada de enlace do IEEE 802.15.4 é responsável por fornecer comunicação entre os dispositivos e se divide em duas subcamadas: MAC (*Medium Access Control*) e LLC (*Logical Link Control*).

1. **Subcamada MAC**: Controla o acesso ao meio de comunicação e gerência a interação com a camada física. Possui dois modos de operação:
   - *Modo Beacon-enabled*: Utiliza estruturas chamadas superframes e beacons para sincronização e transmissão de dados. Os beacons são transmitidos periodicamente para sincronizar os dispositivos na rede. Este modo geralmente tem mais tráfego e latência devido ao envio regular de pacotes de controle.
   - *Modo Beaconless*: Não usa beacons e, em vez disso, emprega o protocolo CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) para controle de acesso ao meio. Os dispositivos operam de forma assíncrona, aguardando transmissões.
2. **Subcamada LLC**: É responsável por fornecer uma interface entre a subcamada MAC e a camada de rede, abstraindo os detalhes da camada física. Isso permite que a camada de rede opere independentemente dos detalhes do meio de transmissão físico.

Além disso, a camada de enlace inclui o suporte para endereçamento curto (16 bits) e longo (64 bits), e também mecanismos de segurança para autenticação e criptografia de dados.

O IEEE 802.15.4 fornece a base para a camada física e de enlace de dados. Porém, para possibilitar a comunicação de dispositivos com redes IP, como a Internet, é necessário um protocolo de adaptação. O 6LoWPAN é um exemplo de protocolo que desempenha este papel, permitindo que pacotes IPv6 sejam enviados e recebidos sobre redes IEEE 802.15.4. Ele otimiza esses pacotes para o tamanho e capacidade das redes IEEE 802.15.4, facilitando a integração de dispositivos de baixo consumo em redes IP.

2.6 6LOWPAN

O 6LowPAMN introduziu o conceito de uma subcamada de adaptação dentro da camada de enlace de dados do modelo OSI, que permite a transmissão de datagramas IPv6 sob enlaces

de rádio IEEE 802.15.4. Ele permite a transmissão de pacotes IPv6 em redes de baixa potência, como as redes de sensores sem fio. O 6LoWPAN utiliza técnicas de compressão e fragmentação para reduzir o tamanho dos pacotes e otimizar a utilização da largura de banda disponível. Dessa forma, ele possibilita a integração de dispositivos IoT em redes IP, garantindo a interoperabilidade e facilitando a comunicação entre os nós (PAPADOPOULOS *et al.*, 2018).

Ao permitir a comunicação de dispositivos de baixa potência com redes IP, o 6LowPan, necessita de um protocolo de roteamento eficiente para gerenciar a transmissão de pacotes em redes de sensores. O RPL foi desenvolvido para atender a essa necessidade, oferecendo um mecanismo de roteamento escalável e adaptável especificamente projetado para redes de baixa potência e com perdas, como as que utilizam 6LoWPAN.

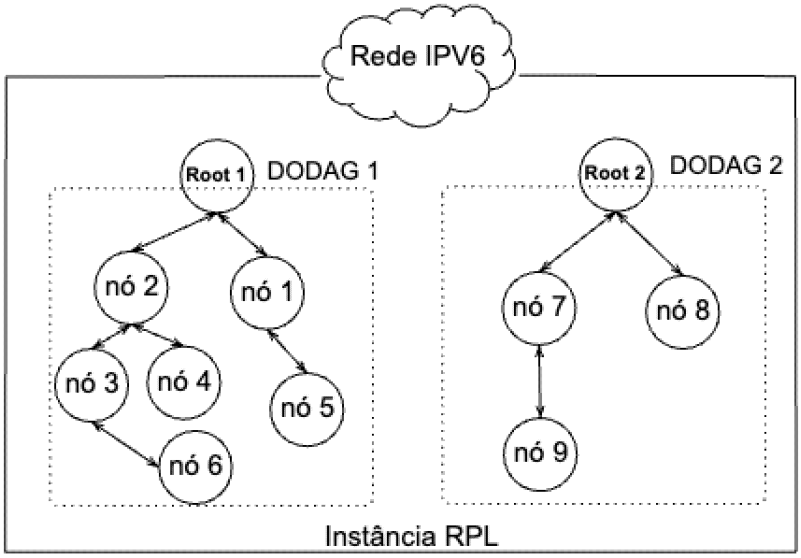
2.7 RPL

O RPL é um protocolo de roteamento de vetor de distância livre de loops projetado para redes sem fio com baixo consumo de energia e suscetíveis a perda de pacotes. Ele adota uma abordagem proativa, encontrando e mantendo rotas em toda a rede, mesmo em condições desafiadoras (DARABKH *et al.*, 2022). Ele estabelece uma estrutura hierárquica de roteamento chamada DODAG (*Destination Oriented Directed Acyclic Graph*), ilustrado na Figura 8, que permite o encaminhamento eficiente dos pacotes entre os nós. O RPL é responsável por determinar nativamente os melhores caminhos de roteamento, levando em consideração métricas como consumo de energia, qualidade do link e latência.

Ao entrar na rede RPL, um novo nó se comunica com os nós existentes para explorar a estrutura da rede e escolher um nó pai, com base em um algoritmo de ranqueamento (utilizando uma função custo). O nó pai geralmente será um nó com uma conexão com menor custo e qualidade de enlace. O nó raiz é o nó no topo da topologia DODAG e é responsável por atribuir endereços IP a novos nós. Quando um nó é adicionado à rede, ele recebe um endereço IP da família do nó raiz, como ilustrado na Figura 9. Este endereço IP permitirá que o novo nó entre na rede e se comunique com outros nós usando o protocolo RPL, projetado para otimizar o encaminhamento de pacotes em redes de sensores sem fio e ambientes de baixo consumo de energia.

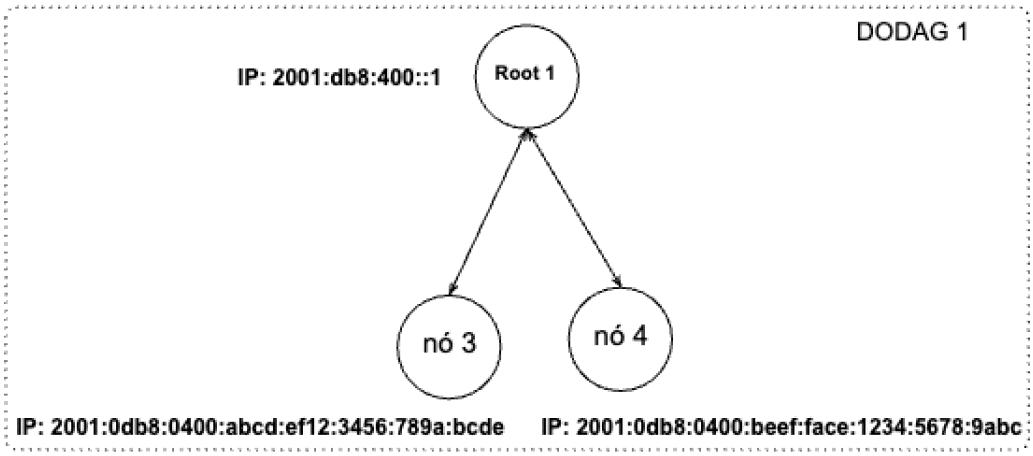
Logo no início de uma instância RPL, quando um *root* é iniciado em uma DODAG, ele começa a transmitir mensagens ICMPv6 chamadas DIO (DODAG Information Object), que são mensagens utilizadas para efetuar o ranqueamento de um nó. Essas mensagens contêm

Figura 8 – Instância RPL



Fonte: Autoria própria (2023).

Figura 9 – Distribuição de IPs em uma instância



Fonte: Autoria própria (2023).

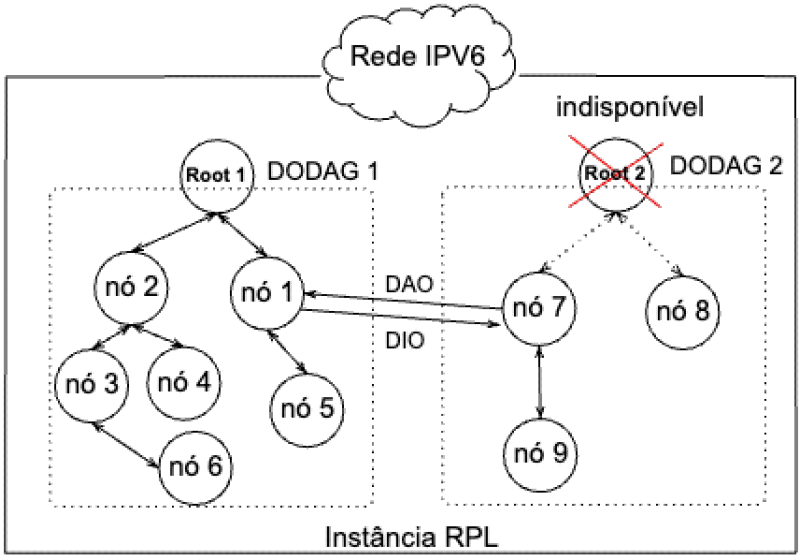
informações sobre a DODAG e a qualidade do enlace de nós vizinhos. Ao receber essa mensagem, um nó calcula o próprio *rank* com base na métrica de roteamento e transmite à frente seu próprio DIO com essa informação para os nós seguintes. Cada nó então seleciona um pai com base no *rank* calculado, formando assim uma rota até o *root* da DODAG (WINTER *et al.*, 2012).

Caso um nó perca a conexão com seu nó pai ou com a raiz da DODAG, seja por qualquer motivo, como seu pai estar indisponível, o nó pode iniciar um processo de reparação local. Esse processo consiste em procurar um novo pai dentro da mesma DODAG. Para isso, o nó transmite uma solicitação DAO para todos os nós vizinhos, que respondem com mensagens DIO. Caso o

nó não consiga estabelecer uma nova rota até a raiz da DODAG, em um determinado período, ele pode iniciar um processo de reparação global. Nesse processo, a raiz do DODAG é informada da falha e inicia um novo processo de formação do DODAG.

Se existir outro nó raiz na instância RPL, um nó pode se reconectar a ela por meio do processo regular de formação do DODAG, ilustrado na Figura 10. A nova raiz transmitirá DIOs, e o nó calculará seu *rank* e selecionará um pai, como antes (WINTER *et al.*, 2012).

Figura 10 – Reparação rede RPL



Fonte: Autoria própria (2023).

É importante destacar um aspecto característico da estrutura de comunicação dentro de um DODAG, ilustrado a cima, os nós se comunicam apenas com o nó raiz, também conhecido como root. Isso significa que, se um nó dentro da rede deseja enviar uma mensagem para outro nó que não seja o nó root , ele não a envia diretamente para o nó de destino. Em vez disso, a mensagem é primeiro enviada para o nó raiz, que atua como uma espécie de centralizador ou intermediário na comunicação. O nó raiz possui informações sobre a estrutura da rede e sabe como encaminhar o pacote para o nó de destino apropriado através do caminho mais eficiente. Esse mecanismo garante que as mensagens sejam encaminhadas de maneira eficaz em redes onde os recursos são limitados.

Após compreender os diversos protocolos essenciais ao estabelecimento e otimização das comunicações em uma rede de sensores, é importante reconhecer que, além dos protocolos, é necessário ter a presença de um sistema operacional no dispositivo final para gerenciar os recursos e possibiltar a implementações de aplicações com regras especificas. O RIOT-OS é um exemplo de sistema operacional desenvolvido com foco em IoT, que se adapta perfeitamente às

redes de sensores.

2.8 SISTEMA OPERACIONAL RIOT-OS

Segundo Bormann, Ersue e Keranen (2014), dispositivos de IoT costumam ter recursos de hardware limitados, no geral dispositivo de baixo custo não tem poder computacional para executar SOs complexos como Linux ou Windows. Assim surge a necessidade de sistemas operacionais dedicados para IoT, mais enxutos e com recursos nativos de melhor desempenho que facilitam na hora do desenvolvimento de aplicações *IoT*. O sistema operacional RIOT é um exemplo de SO desenvolvido especialmente para aplicações IoT, baseado em uma arquitetura modular construída em torno de um kernel minimalista e desenvolvido por uma comunidade mundial de desenvolvedores, escrito para dispositivos de baixo custo. Para sua execução exige memória mínima de 10KB e não necessita que o hardware possua MMU (*Memory Management Unit*) nem MPU (*Memory Protection Unit*) (BACCELLI *et al.*, 2018). Além de necessitar menos memória, o RIOT suporta uma gama grande de arquiteturas de 8 a 32 bits, fornecendo um conjunto completo de recursos de um sistema operacional, desde abstração de hardware, recursos de kernel, bibliotecas de sistema até ferramentas.

Alguns princípios que explicam partes do projeto RIOT são:

- **Padrões de rede:** Se concentra em especificações de protocolo de rede padrão aberto, por exemplo, protocolos IETF (Internet Engineering Task Force);
- **Padrões do sistema:** Cumpre os padrões relevantes, por exemplo, o padrão ANSI C (C99), para aproveitar ao máximo o maior conjunto de softwares de terceiros. O uso da linguagem C atende a requisitos de recursos de baixo nível e fácil programação;
- **APIs unificadas:** Fornece consistência de APIs em todos os hardwares suportados, mesmo para APIs de acesso a hardware, para atender à portabilidade de código e minimizar a duplicação de código;
- **Modularidade:** Define blocos de construção independentes, a serem combinados de todas as maneiras possíveis, para atender a casos de uso versáteis e restrições de memória;
- **Memória estática** - Faz uso extensivo de estruturas pré-alocadas para atender tanto à confiabilidade, validação e verificação simplificadas, quanto aos requisitos em tempo real;
- **Independência de fornecedor e tecnologia** - As bibliotecas de fornecedor são

normalmente evitadas, para diminuir as chances de aprisionamento do fornecedor e minimizar a duplicação de código, além disso, as decisões de design não devem vincular a RIOT a uma tecnologia específica;

- **Comunidade de código aberto aberta e inclusiva** - Permanece livre e aberto para todos e agregar uma comunidade com processos 100% transparentes.

Ao considerar a limitação de recursos dos dispositivos de IoT e a necessidade de sistemas operacionais enxutos, como o RIOT, torna-se evidente a importância de uma arquitetura de alta disponibilidade. Esta arquitetura deve garantir a operação contínua e sem interrupções destes dispositivos, mesmo que ocorram falhas ou interrupções de serviço.

2.9 ARQUITETURA DE ALTA DISPONIBILIDADE

Com a junção de componentes da camada física, processadores e discos, quanto da camada de software, protocolos de comunicação e SOs, temos uma arquitetura formada pela união desses componentes, que trabalham de forma cooperativa. Uma arquitetura de alta disponibilidade é o conjunto de tecnologias em que se tem o mínimo de interrupções e indisponibilidade de serviços de computação, proporcionando serviços redundantes e tolerantes a falhas em um *datacenter*.

Para a maioria das aplicações de IoT, a alta disponibilidade é fundamental. Existem várias formas de atingir essa disponibilidade em um sistema, por exemplo, detecção e recuperação de falhas (YANG; KIM, 2019).

- **Detecção de falhas** - Pode ser feita ao nível físico, virtual e de aplicação, podendo conter um ou mais niveis de aplicação;
- **Recuperação de falhas** - A recuperação de falhas é uma estratégia crucial em sistemas críticos, empregando tolerância a falhas e redundância. A tolerância a falhas mantém o sistema operante, mesmo quando ocorrem problemas, através do reinício do sistema ou redirecionamento do fluxo para outro servidor. A redundância, por sua vez, implica na preparação de um sistema paralelo, que assume em caso de falha, oferecendo uma resposta mais rápida, embora tenha custos maiores.

Um componente essencial dentro das arquiteturas de alta disponibilidade remete à forma como a aplicação é implantada, ou seja, ao modelo de computação adotado.

2.10 MODELOS DE COMPUTAÇÃO

Quando se fala de modelos de computação ou formas de implantar aplicações, tem-se três formas:

- **Nuvem** - Empresa contrata serviços de computação de terceiros para manter suas aplicações e não tem responsabilidade sob manutenção e atualizações dos servidores.
- **On-premise** - Empresa mantém seus servidores em sua estrutura física e responsável por todo o processo de *upgrade* e manutenção.
- **Híbrido** - Empresa mantém parte das aplicações em sua estrutura e outra parte na nuvem.

Nos últimos anos o modelo de computação em nuvem entrou em ascensão por muitos motivos, como: menor custo de operação, não necessitar de time especializado dedicado a cuidar de servidores, economia de espaço e recursos energéticos. Outro ponto muito importante é a elasticidade provida pelo modelo de computação em nuvem, que é muito facil de redimensionar a memória e processar nos recursos de computução.

2.10.1 Computação em nuvem

Com a expansão tecnológica em todos os setores, teve uma alta na demanda por processamento em nuvem, devido às mudanças de dimensão serem mais rápidas e a elasticidade provida pela computação em nuvem ajudar empresas a dimensionar o poder computacional necessário para soluções de forma mais simples e rápida. No entanto, enquanto as tecnologias de nuvem trazem recursos e serviços de alto desempenho para empresas com melhor disponibilidade e preço, muitos aplicativos exigem uma forte necessidade de elasticidade, escalabilidade de recursos e baixa latência na transmissão de dados (BAKHOUYA *et al.*, 2020).

2.10.2 Computação de borda

Computação de borda é o nome dado a processos de computação que são executados próximos à fonte de dados, segundo Cao *et al.* (2020), os dados são coletados por dispositivos e passam por um processamento próximo a si, antes de serem enviados a nuvem.

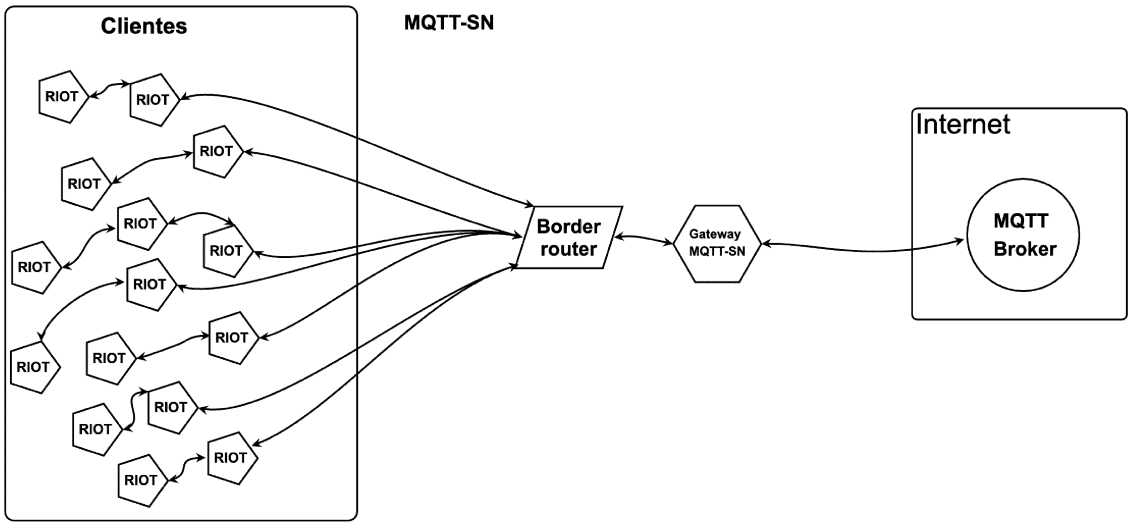
Para Satyanarayanan (2017), computação de borda é definido como a execução de recursos na borda da rede mais próxima de dispositivos ou sensores móveis. É um novo modelo

de computação que une recursos que estão próximos ao usuário em distância geográfica ou distância de rede para fornecer serviços de computação, armazenamento e rede para aplicativos. O principal benefício da computação de borda é a redução da latência, uma vez que os dados não precisam ser enviados a um *datacenter* remoto para processamento. Isso é particularmente importante para aplicações de tempo real ou que necessitam de respostas rápidas.

3 METODOLOGIA DE DESENVOLVIMENTO

A Figura 11 mostra uma rede de sensores convencional com apenas um border router e um *gateway* MQTT-SN. Embora seja uma configuração adequada, ela apresenta uma limitação significativa. Ao possuir um único roteador de borda e *gateway*, o sistema fica suscetível a um ponto de falha único. Ou seja, qualquer problema que ocorra no *border router* ou no *gateway* pode comprometer todo o funcionamento do sistema. É importante destacar que essa vulnerabilidade crítica representa um risco para a operação eficiente e confiável da rede.

Figura 11 – Rede de sensores convencional



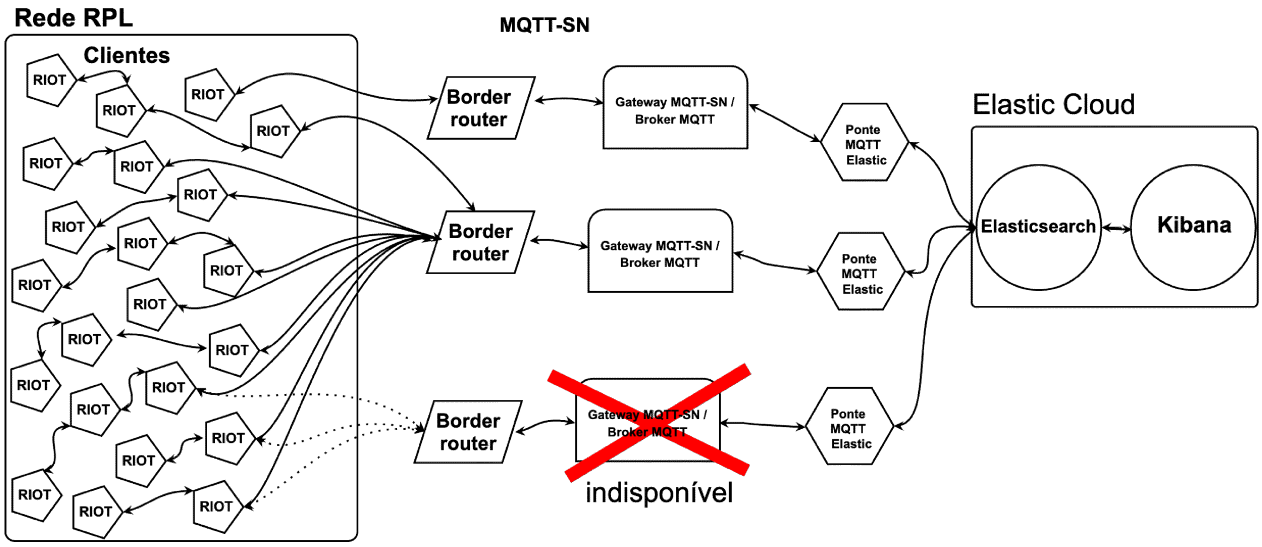
Fonte: Autoria própria (2023).

Para tratar essa problemática de disponibilidade, propõe-se a implantação de uma arquitetura multi *border router* e *multi-gateway*, conforme a Figura 12. Isso envolve a criação de uma rede de sensores em formato de malha, na qual se utiliza o protocolo MQTT-SN, sobre os protocolos 6LowPan e RPL para comunicação entre os nós e definição de custo e caminho de rotas entre os nós e o nó principal. Nessa abordagem, pode-se ter N gateways, onde cada *gateway* será um nó *root* da rede RPL. A arquitetura proposta é genérica e poderia ser aplicada a diversos contextos do dia a dia e em cidades inteligentes. No trabalho, realiza-se uma aplicação que coleta dados de sensores de temperatura, umidade, acelerômetro e magnetômetro; após coletar esses dados, eles são enviados ao *gateway* MQTT-SN, onde ocorre a ponte desses dados com a Internet.

A arquitetura proposta utiliza uma série de protocolos para garantir a comunicação eficiente e confiável entre os nós da rede. Cada um desses protocolos opera em uma camada específica do modelo OSI, conforme ilustrado na Figura 13. Esses protocolos trabalham juntos

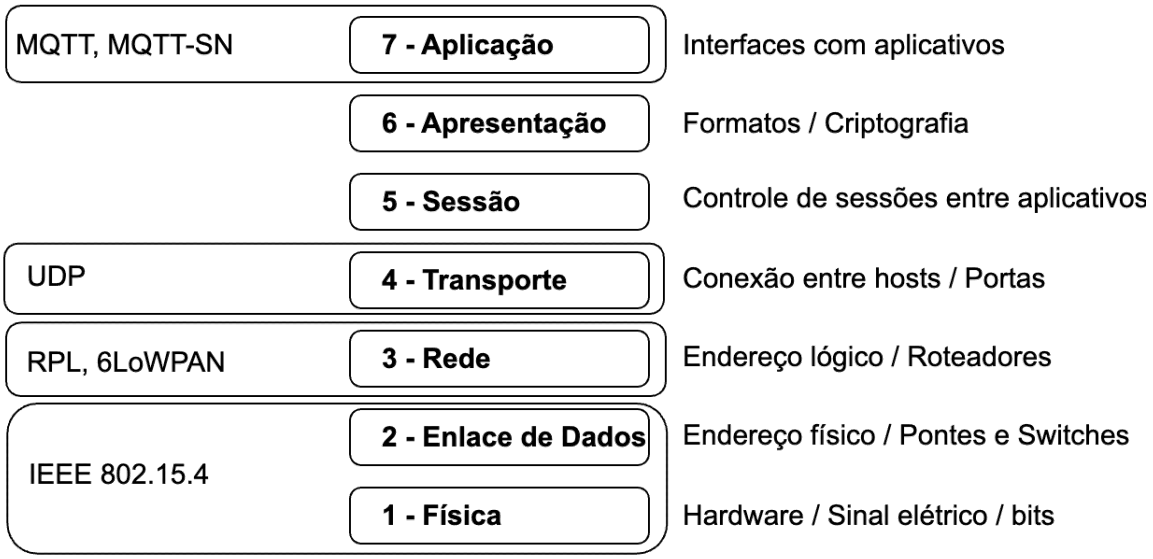
para garantir que os dados coletados pelos sensores sejam transmitidos de maneira eficiente e confiável.

Figura 12 – Rede de sensores *multi-gateway*



Fonte: Autoria própria (2023).

Figura 13 – Protocolos da arquitetura



Fonte: Autoria própria (2023).

Este capítulo apresenta a metodologia de desenvolvimento do trabalho proposto, iniciando com a configuração do ambiente de desenvolvimento. Posteriormente, com o ambiente configurado, inicia-se o desenvolvimento de *firmware* utilizando o sistema operacional RIOT para os dispositivos presentes na rede de sensores implantada. Com o *firmware* desenvolvido se

dá início a implantação da rede de sensores em que são utilizados *gateways* para o protocolo MQTT-SN fazendo a ponte com o protocolo MQTT. Após ter uma rede de sensores funcionando e comunicando com um broker MQTT, é simulado cenários de desbalanceamento de carga entre clientes e broker MQTT de modo a configurar balanceamento de carga para esses cenários. Por fim se tem o envio dos dados do broker para a nuvem e validação da arquitetura em sua totalidade.

3.1 PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para começar a desenvolver a arquitetura proposta é necessário preparar o ambiente de desenvolvimento. Para isso um conjunto de ferramentas serão configuradas e instaladas. As ferramentas utilizadas são plataformas hospedadas em nuvem e softwares de código aberto, dentre elas:

- FIT IoT-LAB;
- Visual Studio Code v1.73;
- Golang v1.20
- Elasticsearch V8.8
- Ubuntu v22.04.2 LTS;
- Mosquitto Really Small Message Broker
- MQTT-SN v1;
- MQTT v3.1.1;
- RIOT-OS 2023.04;

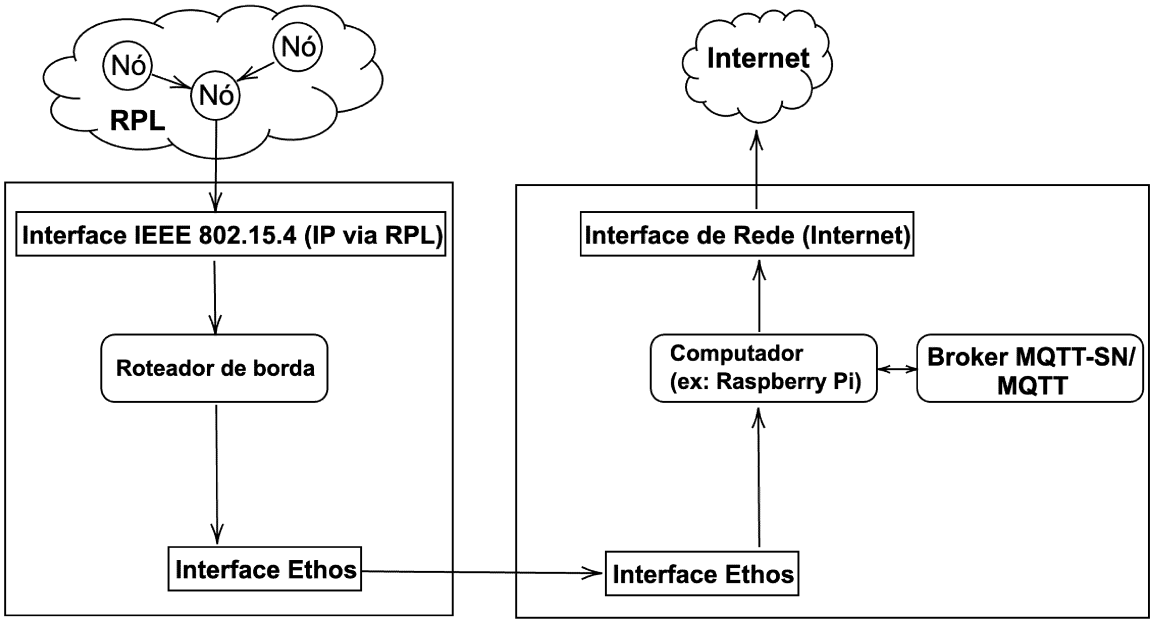
3.2 DESENVOLVIMENTO FIRMWARE RIOT-OS

Após a instalação e configuração de todos os softwares listados acima, inicia-se o desenvolvimento do firmware para a aplicação IoT. Para isso, utiliza-se o sistema operacional RIOT-OS e a linguagem de programação C. Na arquitetura, estão presentes duas aplicações: o roteador de borda, que faz a tradução dos dados que chegam via 6LowPan para o gateway MQTT-SN, e os clientes finais, que constituem a rede de sensores e enviam telemetria.

3.2.1 Roteador de borda

Um roteador de borda (*border router*), é um dispositivo que conecta uma rede local (LAN) a redes externas, como a Internet. No exemplo da arquitetura, o roteador de borda fará a ponte entre o 6LoWPAN proveniente da rede RPL e o gateway MQTT-SN, o qual já possui acesso à internet. A estrutura da conexão entre o roteador de borda e o gateway pode ser observada na Figura 14. O roteador de borda possui uma interface IP para comunicação via 6LoWPAN com a rede RPL e uma interface Ethos, utilizada para conectar-se a outro computador. Este último possui duas interfaces: uma Ethos para receber dados do roteador de borda e outra que é uma interface de rede conectada à Internet.

Figura 14 – Estrutura roteador de borda



Fonte: Autoria própria (2023).

Porém, para que funcionasse de forma adequada na arquitetura proposta, é necessário configurar e desenvolver alguns componentes, sendo eles: rede RPL, roteador de borda, servidor *socket*, configuração do root node da rede RPL.

3.2.1.1 Firmeware base

No início do desenvolvimento do firmware do roteador de borda, o Makefile do projeto deve ser modificado, adicionando-se os módulos necessários para a configuração da rede RPL e para a criação do servidor UDP.A seguir, são apresentados os trechos do Makefile modificado

para ilustrar as alterações realizadas, as quais podem ser visualizadas no Código-fonte 1.

Código-fonte 1 – Alterações Makefile RIOT-OS

```
1  APPLICATION = gnrc_border_router_rpl
2  ...
3  CFLAGS += –DGATEWAY_IPV6_ADDR=\"2001:660:3207:400::64\"
4  CFLAGS += –DROOT_IPV6_ADDR=\"2001:db8::1\"
5  ...
6  USEMODULE += gnrc_sixlowpan_border_router_default
7  USEMODULE += gnrc_rpl
8  USEMODULE += auto_init_gnrc_rpl
9  ...
10 CFLAGS += –DCONFIG_GNRC_IPV6_NIB_NUMOF=32
11 CFLAGS += –DCONFIG_GNRC_IPV6_NIB_OFFL_NUMOF=32
12 ...
13 USEMODULE += gnrc_icmpv6_echo
14 USEMODULE += shell_cmd_gnrc_udp
15 ...
16 USEMODULE += posix_sockets
17 USEMODULE += sock_udp
18 USEMODULE += posix_sleep
19 USEMODULE += posix_inet
20 USEMODULE += shell
21 USEMODULE += shell_cmds_default
22 USEMODULE += ps
23 USEMODULE += netstats_l2
24 USEMODULE += netstats_ipv6
25 USEMODULE += netstats_rpl
26 ...
```

Autoria própria (2023)

É importante entender o que cada uma dessas flags e módulos representa. Elas são essenciais para a configuração e o funcionamento correto do roteador de borda e da rede RPL. Cada uma delas tem o seguinte objetivo:

- `CFLAGS += -DGATEWAY_IPV6_ADDR=¨2001:660:3207:400::64¨`: Aqui é definido o endereço IPv6 do Gateway MQTT-SN.
- `CFLAGS += -DROOT_IPV6_ADDR=¨2001:db8::1¨`: Este é o endereço IPv6 global para a raiz da rede RPL.
- `USEMODULE += gnrc_sixlowpan_border_router_default`: Este módulo é essencial para o funcionamento do roteador de borda 6LoWPAN.
- `USEMODULE += gnrc_rpl`: Este módulo adiciona o protocolo de roteamento RPL.
- `USEMODULE += auto_init_gnrc_rpl`: Esta flag serve para a inicialização automática do RPL.

- `CFLAGS += -DCONFIG_GNRC_IPV6_NIB_NUMOF=32` e `CFLAGS += -DCONFIG_GNRC_IPV6_NIB_OFFL_NUMOF=32`: Estas flags configuram o tamanho do Neighbor Information Base (NIB) do gnrc_ipv6. Esse espaço de armazenamento precisa ser grande o suficiente para armazenar informações para cada nó na rede.
- `USEMODULE += gnrc_icmpv6_echo`: Este módulo permite ao roteador enviar e receber mensagens ICMPv6 Echo (ping).
- `USEMODULE += shell_cmd_gnrc_udp`: Este módulo habilita comandos UDP no shell.
- `USEMODULE += posix_sockets`: Este módulo permite o uso de sockets POSIX.
- `USEMODULE += sock_udp`: Habilita a utilização de UDP sockets.
- `USEMODULE += posix_sleep`: Este módulo permite o uso da função sleep POSIX.
- `USEMODULE += posix_inet`: Este módulo permite o uso de funções de internet POSIX.
- `USEMODULE += shell`: Este módulo habilita o shell interativo.
- `USEMODULE += shell_cmds_default`: Adiciona os comandos padrões para o shell.
- `USEMODULE += ps`: Este módulo habilita a listagem de threads ativas.
- `USEMODULE += netstats_l2`, `USEMODULE += netstats_ipv6` e `USEMODULE += netstats_rpl`: Esses módulos fornecem estatísticas sobre a rede no nível 2, IPv6 e RPL, respectivamente.

3.2.1.2 Rede RPL

Por padrão, em uma rede de sensores, os dispositivos são geralmente configurados em modo estrela, onde cada dispositivo da rede se conecta a um único ponto central, o roteador de borda. Essa configuração pode limitar a rede em diversos aspectos, como alcance, devido à necessidade de cada sensor se comunicar diretamente com o roteador de borda, e também resiliência, pois se o ponto central falhar, a rede inteira fica inoperante.

Para melhorar o alcance e a resiliência da rede, iremos utilizar uma topologia de rede em árvore utilizando o protocolo RPL. Neste esquema, os dispositivos estão interconectados, de modo que cada um pode se comunicar com vários vizinhos, e não apenas com um ponto central. Cada roteador de borda presente na arquitetura é um tronco de uma árvore e os outros nós são galhos e folhas, que podem atuar somente como dispositivos finais (folhas) ou roteadores

(galhos) para outros dispositivos. Essa configuração permite que os dados percorram diversos caminhos para alcançar seu destino, o que facilita a manutenção da comunicação mesmo se alguns dispositivos falharem.

Na topologia em árvore, os sensores podem atuar como retransmissores, encaminhando os dados de outros sensores para o roteador de borda. Isso significa que um sensor que está muito distante do roteador de borda para se comunicar diretamente com ele pode, ainda assim, enviar dados por meio de outros sensores que estão mais próximos. Essa retransmissão de dados mediante múltiplos saltos aumenta significativamente o alcance da rede.

Além disso, uma rede em árvore é capaz de reconfigurar-se automaticamente. Se um dispositivo falhar ou for removido, os outros dispositivos podem se reorganizar para estabelecer novas rotas de comunicação. Isso torna a rede mais resistente a falhas.

Após adicionar as configurações do RPL no Makefile, é necessário inicializar a rede RPL no roteador de borda. Isso é realizado através da API do RPL e da interface de rede do RIOT, utilizando a função *gnrc_rpl_init*. Esta função é responsável por iniciar a rede no nó.

3.2.1.3 Nó root rede RPL

Com a rede já em operação, o próximo passo é a configuração de um endereço IP para atuar como root na rede. Para isso, é utilizada a função *gnrc_rpl_root_init*, que define o endereço IP como *root* e retorna uma instância de RPL.

Um ponto de atenção após a configuração do nó raiz, nesse caso o roteador de borda, é estabelecido um ponto central a partir do qual a rede RPL é organizada. Todo cliente que se conectar à rede e estiver na árvore de um nó raiz receberá um IP adjacente ao nó raiz, ou seja, IPs que estão próximos na hierarquia de roteamento da rede. Essa configuração permite uma comunicação eficiente entre os dispositivos na rede RPL. Os clientes podem se comunicar diretamente com o nó raiz ou com outros dispositivos próximos a eles na hierarquia de roteamento, o que reduz a latência e melhora a eficiência da comunicação. Ao atribuir IPs adjacentes aos clientes, a rede RPL estabelece uma estrutura organizada e otimizada para o roteamento de dados. Isso facilita a identificação e o encaminhamento de pacotes na rede, permitindo uma comunicação eficaz entre os dispositivos conectados.

3.2.1.4 Servidor socket

Ao implementar a rede utilizando RPL, nos deparamos com um cenário em que os clientes finais não conseguiam determinar o endereço IP que *gateway* MQTT-SN estava configurado. Para resolver esse problema, inovamos ao planejar e implementar um servidor *socket* no roteador de borda. Esse servidor *socket* é responsável por escutar solicitações UDP provenientes de clientes finais na rede RPL e responder-lhes com o endereço em que o gateway está em execução. Esta abordagem foi uma inovação desenvolvida especificamente para este projeto, e não era uma solução previamente existente.

Para isso, é criada uma nova *thread* na aplicação, utilizando a API de *sockets* e UDP. Durante esse processo, um handler é implementado para aguardar e processar solicitações que seguem um padrão específico. Assim que uma nova solicitação é recebida, o servidor UDP a valida e a processa. Caso essa solicitação seja para obter informações do IP do *gateway*, o servidor UDP responde com o endereço IPv6 correspondente como resposta.

A implementação do servidor consiste em um laço infinito que fica ouvindo e respondendo às solicitações UDP. Cada solicitação é validada se for do tipo `gateway_ipv6_request` e, se este for o caso, o servidor responde à solicitação com o IPv6 do gateway MQTT-SN conectado a ele. O código implementado para realizar essa tarefa é apresentado no Código-fonte 2:

Como foi utilizado o protocolo IEEE 802.15.4 na rede de sensores, o nó *root*, que atua como um dispositivo centralizador, desempenha um papel crítico na comunicação com a internet. É importante destacar que o IEEE 802.15.4 é um padrão para *low-rate wireless personal area networks* (LR-WPANs), e possui uma taxa de transmissão relativamente baixa de 250 kbit/s. Dado que o nó *root* centraliza o tráfego de dados, a largura de banda total disponível para acesso à internet em todos os nós fica limitada a esta taxa de 250 kbit/s. No entanto, isso não é um problema nesse cenário de redes de sensores, pois geralmente esses dispositivos lidam com quantidades relativamente pequenas de dados. Os sensores frequentemente transmitem leituras como temperatura, umidade ou outros dados ambientais que não requerem uma largura de banda significativa. Além disso, as transmissões podem não ser contínuas e são muitas vezes espaçadas ao longo do tempo, o que torna a taxa de 250 kbit/s adequada para esses tipos de aplicações. O foco principal em redes de sensores é muitas vezes a eficiência energética e a capacidade de comunicação em áreas extensas ou desafiadoras, em vez de alta velocidade de transmissão de dados.

Além disso, as redes baseadas no protocolo IEEE 802.15.4 são suscetíveis a perdas

Código-fonte 2 – Configuração Servidor UDP

```
while (1)
{
    sock_udp_ep_t remote;
    ssize_t res;

    if ((res = sock_udp_recv(&sock, buf, sizeof(buf),
            SOCK_NO_TIMEOUT, &remote)) >= 0)
    {
        puts("Received a message");
        /* Check if the message is "gateway_ipv6_request" */
        if (strncmp((char *)buf, "gateway_ipv6_request", 20) == 0)
        {
            if (sock_udp_send(&sock, GATEWAY_IPV6_ADDR,
                    strlen(GATEWAY_IPV6_ADDR), &remote) < 0)
            {
                puts("Error sending reply");
            }
        }
    }
}
```

Autoria própria (2023)

de pacotes devido a diversos fatores, como interferências, congestionamento e limitações no alcance dos rádios. Devido a isso, o uso de protocolos de comunicação como o TCP, que depende de confirmações e retransmissões, pode ser ineficiente e até mesmo prejudicial. As tentativas de retransmissão e os mecanismos de controle de congestionamento do TCP podem não ser adequados para as condições dessa rede, tornando necessário o uso de protocolos mais leves e adaptados a ambientes com baixa taxa de transmissão e propensos a perdas de pacotes.

Em resumo, o desenvolvimento para o roteador de borda garante uma configuração e inicialização eficazes da rede RPL, estabelecimento de um endereço IP válido como *root*, além da inicialização e configuração de um servidor UDP cuja função é ouvir solicitações e retornar a informação do endereço IP do gateway MQTT-SN.

3.2.2 Clientes finais

A aplicação é baseada em um exemplo fornecido pela comunidade, que consiste em uma implementação leve e altamente configurável do MQTT-SN para o sistema operacional RIOT OS. O cliente "emcute mqttsn"no RIOT OS é uma aplicação de exemplo que demonstra como utilizar o Emcute para se conectar a um broker MQTT-SN e estabelecer comunicação com

ele. A partir desse exemplo, foram realizadas uma série de modificações. Entre elas, todas as configurações presentes no Makefile do firmware do roteador de borda também foram aplicadas ao projeto do cliente. Essa configuração permite que o cliente seja incorporado na rede RPL e se comunique com o roteador de borda por meio do protocolo UDP. Dessa forma, é possível estabelecer a comunicação entre o cliente e o roteador de borda na rede RPL.

Além disso, na aplicação, é realizada a configuração via código para inicializar a rede RPL e fazer com que o cliente se junte a ela. Logo após a inicialização, o cliente verifica se existe um nó pai ao qual esteja conectado. Caso haja um nó pai disponível, é possível obter o prefixo do endereço IP desse nó para realizar uma requisição UDP ao nó raiz, solicitando o endereço onde o *gateway* está executando.

A obtenção do prefixo do endereço IP do nó pai é possível porque nós que estão na rede recebem um IP adjacente ao IP do nó raiz. Isso permite que o cliente deduza o endereço do nó raiz e estabeleça uma comunicação com ele. Essa informação é essencial para que o cliente saiba onde está localizado o *gateway* na rede, para que na sequência possa estabelecer uma conexão com ele e possa se comunicar de forma adequadamente.

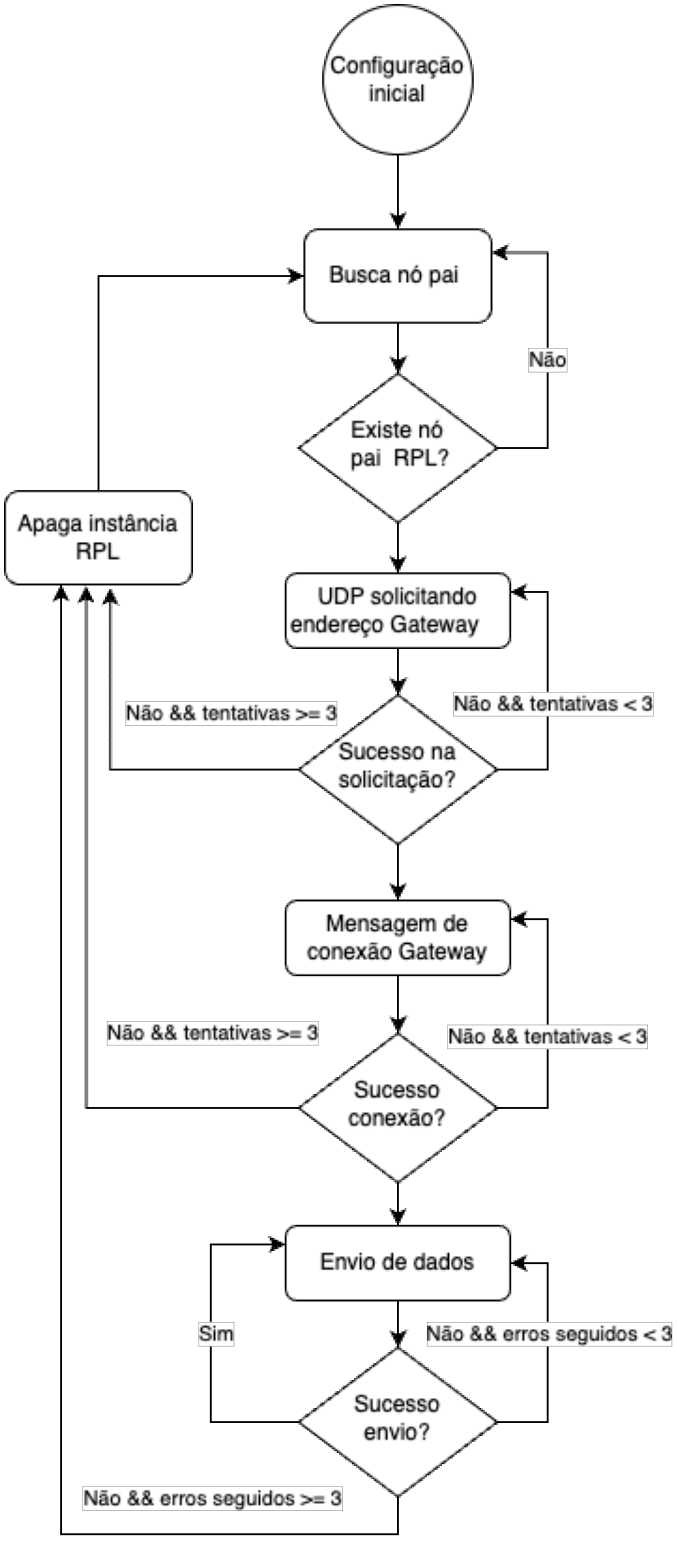
Logo após obter o endereço do roteador de borda e solicitar o endereço do *gateway*, caso consiga o endereço de onde está configurado, o cliente estabelece conexão com o mesmo e começa a enviar periodicamente os dados de telemetria. Caso o cliente não consiga estabelecer a conexão ou se conseguir conectar, mas falhar no envio de dados por três ciclos consecutivos, a instância de RPL do cliente é removida automaticamente e reiniciada. Isso permite que o algoritmo escolha uma nova melhor rota e um novo nó raiz para estabelecer a conexão. Em seguida, o processo de solicitação do endereço do *gateway* via UDP e o estabelecimento da conexão são reiniciados, garantindo, assim, resiliência ao sistema. O diagrama desse processo é ilustrado na Figura 15.

3.3 GATEWAYS MQTT-SN

É essencial entender a conexão entre o nó *root* e o *gateway* para compreender a arquitetura da rede. O *border router*, que desempenha um papel fundamental nesta conexão, é um dispositivo equipado com um microcontrolador e um rádio IEEE 802.15.4. Este dispositivo é responsável por gerenciar uma rede RPL, utilizada para conectar os outros microcontroladores na rede em malha.

O microcontrolador dentro do *border router* utiliza uma interface de rede virtual para

Figura 15 – Lógica implementação clientes RIOT



Fonte: Autoria própria (2023).

estabelecer comunicação com outro dispositivo. Essa interface é implementada por Ethos, que permite que a comunicação de rede seja realizada sobre uma linha serial. Isso é particularmente útil para a comunicação entre o microcontrolador e sistemas externos.

O computador, que está interconectado ao *border router* através da linha serial, executa um software que implementa um gateway MQTT-SN. O gateway faz a ponte entre a rede de sensores gerenciada pelo *border router* e o protocolo MQTT, permitindo que os dispositivos na rede de sensores se comuniquem com a Internet ou outras redes utilizando o protocolo MQTT.

O *gateway* MQTT-SN utilizado para o projeto é o Mosquitto Really Small Message Broker (RSMB), por ser uma implementação de servidor dos protocolos MQTT e MQTT-SN. Qualquer cliente que implemente corretamente este protocolo pode usar este servidor para enviar e receber mensagens. O servidor tem como finalidade receber os dados da rede de sensores e fazer a ponte entre os protocolos MQTT-SN e MQTT. Além disso, também suporta diferentes níveis de Qualidade de Serviço (QoS), permitindo garantir a entrega confiável de mensagens, inclusive em redes instáveis.

Para fazer seu uso, basta baixá-lo, seu código-fonte é aberto e está disponível no Github, instalar e utilizar. Logo após instalar, é necessário criar um arquivo de configuração com o seguinte conteúdo do Código-fonte 3.

Código-fonte 3 – Arquivo Configurações do Gateway

```
1  # add some debug output
2  trace_output protocol
3
4  # listen for MQTT-SN traffic on UDP port 1885
5  listener 1885 INADDR_ANY mqtts
6    ipv6 true
7
8  # listen to MQTT connections on tcp port 1886
9  listener 1886 INADDR_ANY
10   ipv6 true
```

Autoria própria (2023)

Esses dados configuram como o *Broker* irá operar e em quais portas o protocolo MQTT e MQTT-SN serão expostos para que outros serviços possam interagir. Após a instalação do RSMB e a criação do arquivo de configuração, para executar o *Broker*, basta utilizar o seguinte comando `broker_mqtts nome-arquivo.conf`.

Isso iniciará o *Broker* MQTT na porta 1886 e o MQTT-SN na porta 1885, permitindo a comunicação através desses protocolos no endereço IP da máquina onde o *Broker* está sendo executado. Dessa forma, o serviço estará disponível para receber conexões MQTT e MQTT-SN e processar as mensagens enviadas pelos clientes.

3.4 ELASTIC CLOUD

Todos os dados gerados pela aplicação e publicados em um tópico do Broker via MQTT-SN estão disponíveis também para uma subscrição via MQTT. Para armazenar esses dados para análises futuras, foi escolhido como banco de dados o Elasticsearch juntamente com o ELK stack, que possibilita criar visualizações e fazer análises dos dados por meio do Kibana. Para uma maior eficiência e escalabilidade desses serviços na composição da arquitetura do projeto, foi escolhido o serviço em nuvem Elastic Cloud, onde toda a parte de gestão de infraestrutura é vendida como serviço.

O Elastic Cloud é um serviço em nuvem gerenciado que abriga produtos como Elasticsearch e Kibana. O Elasticsearch é um sistema de busca e análise distribuído que lida com grandes volumes de dados rapidamente. Kibana é a interface visual que permite explorar esses dados no Elasticsearch. Na arquitetura proposta, o Elasticsearch é responsável por armazenar e indexar os dados de maneira eficiente, enquanto o Kibana auxilia na visualização e análise desses dados. Um script é utilizado para formar uma ponte que encaminha os dados publicados em um tópico MQTT diretamente para o Elasticsearch.

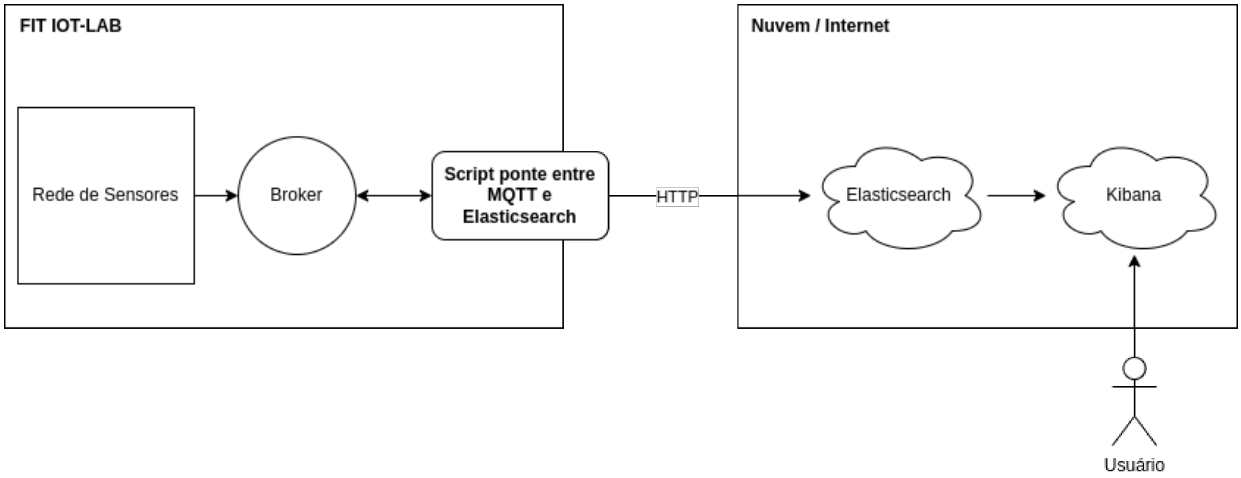
3.5 PONTE MQTT PARA ELASTIC CLOUD

Para coletar dados publicados em um tópico MQTT e enviá-los para um índice no Elasticsearch, foi desenvolvido um script em Golang. O script utiliza as bibliotecas ‘github.com/eclipse/paho.mqtt.golang‘ (v1.4.2) para conectar e interagir com o broker MQTT e ‘github.com/elastic/go-elasticsearch/v8‘ (v8.8.0) para interagir com o Elasticsearch.

A escolha pelo uso da linguagem Golang foi motivada pelas suas características, como a capacidade de gerar um binário executável que pode ser executado em uma variedade grande de dispositivos, proporcionando portabilidade e facilitando a implementação em diversas plataformas. Além disso, Golang é conhecido por sua eficiência em operações de entrada e saída (IO), um requisito importante para a aplicação em questão, que exige alto desempenho na coleta, processamento e transmissão de dados.

O script em Golang é fundamental para ser possível disponibilizar os dados gerados pelos dispositivos na rede de sensores diretamente na nuvem, indexados de forma eficiente no Elasticsearch, para posterior visualização e análise no Kibana. A Figura 16. ilustra o funcionamento desse processo.

Figura 16 – Script ponte entre MQTT e Elastic Cloud



Fonte: Autoria própria (2023).

3.6 CONFIGURAÇÕES DE REDE

Para a configuração da rede, o RPL é configurado através das aplicações RIOT do roteador de borda (*border router*) e do cliente final. Adicionalmente, ao iniciar o roteador de borda, um *script* fornecido pelo RIOT OS é utilizado para configurar uma interface de rede TAP (Tun/Tap). Este *script* cria a interface TAP, em seguida define as rotas de rede utilizando o protocolo UHCPD (*micro Host Configuration Protocol Daemon*). Finalmente, o ethos, um driver de terminal serial, é iniciado na interface, permitindo a comunicação com o dispositivo em que a aplicação do border router está em execução. Se o *script* for interrompido, uma função de limpeza é acionada para remover a interface e reverter as alterações de roteamento feitas.

Devido à configuração especializada feita para a rede operar com RPL, os clientes não conseguem se conectar diretamente ao *gateway*. Por isso, foi necessário estabelecer uma rota entre o roteador de borda e o *gateway*. Essa rota foi configurada adicionando o IP global do nó *root* da rede RPL, o que possibilita que os IPs dos nós, por serem adjacentes, tenham um caminho de comunicação até o *gateway*.

3.7 IMPLANTAÇÃO DA REDE DE SENSORES

A implantação de dispositivos e redes de sensores é realizada por meio da plataforma FIT IoT-LAB. Esta é uma plataforma dedicada a testes de redes heterogêneas de dispositivos e sensores sem fio. Com ela, é possível programar e controlar dispositivos. Importante ressaltar que a plataforma IoT Lab suporta diversos sistemas operacionais, ampliando ainda mais suas

possibilidades de uso e compatibilidade.

A plataforma abriga mais de 1500 nós com diferentes modelos de placas e, desde 2012, já finalizou mais de 370.000 experimentos. Ela permite aos usuários construir aplicações sem qualquer requisito de dependência. Isso pode ser feito do zero, com base em bibliotecas de código aberto, ou mesmo usando um sistema operacional. Esta plataforma é amplamente utilizada pela comunidade de IoT em todo o mundo (IOT-LAB, 2022).

O primeiro passo para implantar um novo experimento nesta plataforma é fazer o upload de um firmware previamente desenvolvido. Após o upload, deve-se selecionar em qual hardware o firmware será implantado, quantas instâncias serão usadas, por quanto tempo e em qual localidade se deseja executar os dispositivos.

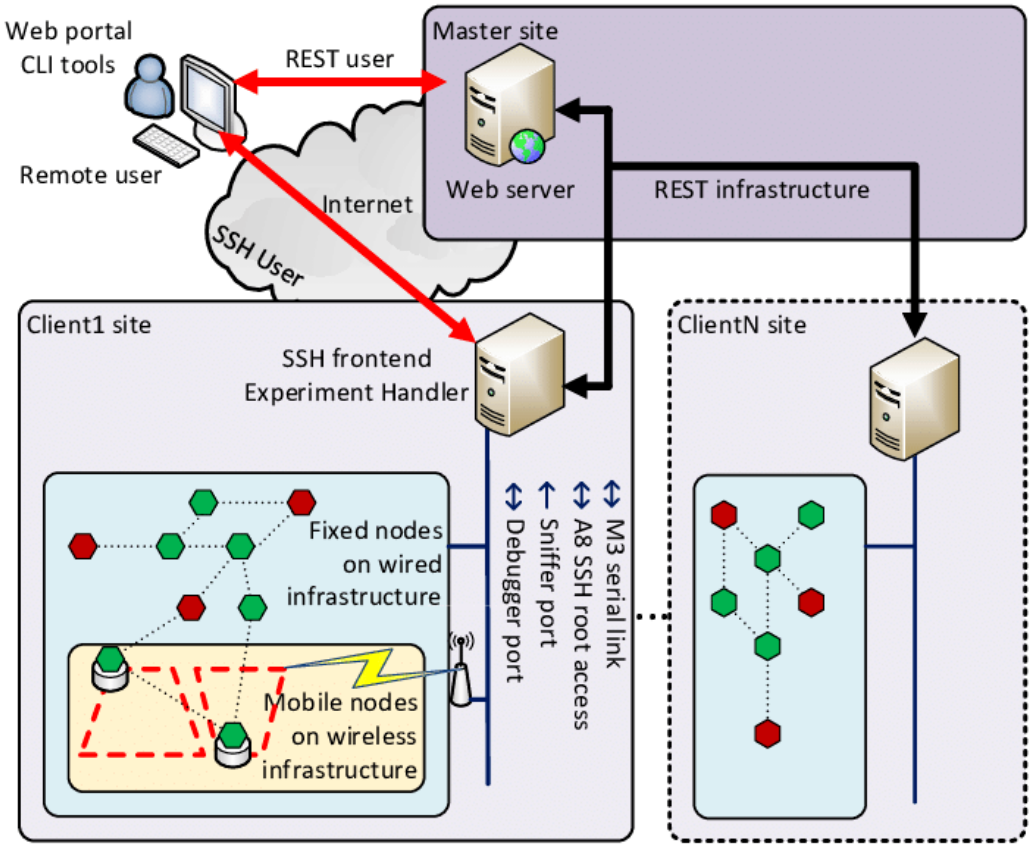
A plataforma IoT Lab não apenas suporta uma vasta gama de dispositivos, mas também possibilita o trabalho e teste com vários protocolos de comunicação, incluindo 802.15.4, Sub-GHz, Bluetooth Low Energy (BLE), LoRa, UWB, entre outros. Para facilitar a usabilidade, oferece acesso remoto, permitindo ao usuário executar operações como flash nos dispositivos, reset e pause. Essa comunicação pode ser realizada por meio de TCP, SSH ou protocolos de websocket.

Além disso, a plataforma oferece a capacidade de monitorar o consumo de energia individual dos dispositivos e analisar o protocolo de rede com ferramentas de sniffer de pacotes. Todas essas funções podem ser acessadas por meio de uma plataforma web ou via linha de comando (CLI) diretamente no servidor SSH. Vale ressaltar que, embora o portal web seja bastante útil, ele pode apresentar mais limitações em relação ao acesso via SSH. A organização e o funcionamento da plataforma FIT IoT-LAB podem ser melhor compreendidos com a ajuda da Figura 17 onde é possível visualizar os componentes, facilitando o entendimento do fluxo de trabalho e das operações realizáveis.

3.8 VALIDAÇÃO DA ARQUITETURA

Para validar a arquitetura são realizados vários testes de caso, começando pelo envio de dados simples dos dispositivos da rede de sensores via *gateway* chegando até o broker MQTT. Após isso serão realizados testes em cenários com indisponibilidade de um *border router* e *gateway* MQTT-SN, visando que o sistema faça um rebalanceamento do tráfego do *gateway* indisponível para um novo *gateway*. Por último, o teste de saturação da rede, em que se tem um *gateway* sobrecarregado e a rede de sensores deve redirecionar parte do tráfego para outro *border*

Figura 17 – Infraestrutura IOT-LAB



Fonte: (HADJ *et al.*, 2015).

router próximo. Para ambos os casos, saturação e indisponibilidade a aplicação deverá mudar automaticamente o nó *root* da rede RPL em que está para um novo e automaticamente comece a enviar os dados para ele.

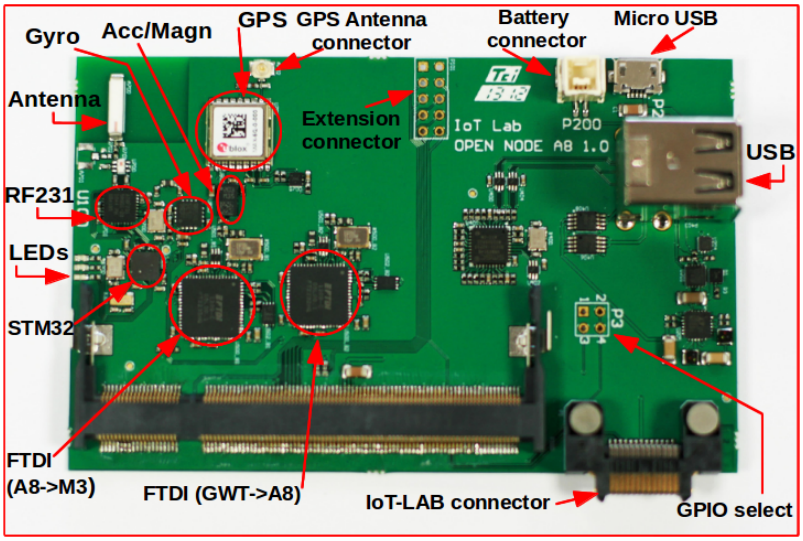
4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados obtidos neste trabalho, além das experiências do autor durante a pesquisa e o desenvolvimento da arquitetura de IoT de alta disponibilidade. Inicialmente, serão apresentados os resultados que demonstram o funcionamento da arquitetura em sua totalidade. Em seguida, será exibido o comportamento da arquitetura em cenários de indisponibilidade de um *gateway*, visando redistribuir a carga do *gateway* indisponível. Posteriormente, será mostrada a integração dos dados provenientes dos clientes com a nuvem e a visualização destes nas dashboards na web por meio do Kibana. Para finalizar, haverá uma breve discussão sobre as vantagens e desvantagens da arquitetura.

4.1 IMPLANTAÇÃO DA REDE DE SENSORES

O principal aspecto desta arquitetura proposta é a rede de sensores. A ideia é que essa arquitetura possa ser reproduzida com qualquer placa que tenha suporte ao sistema RIOT-OS e rádio 802.15.4 para comunicação via 6LowPan. Para a implantação, foram utilizados 120 nós A8-M3 na plataforma IoT-LAB, uma placa não comercial e desenvolvida especificamente para a plataforma. A arquitetura pode ser visualizada em mais detalhes na Figura 18.

Figura 18 – Arquitetura placa A8-M3



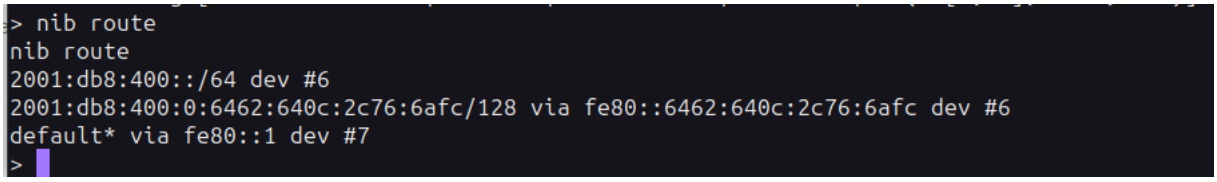
Fonte: (IOT-LAB, 2022).

Para iniciar o experimento na plataforma IoT-LAB, utilizou-se a interface de linha de comando diretamente no servidor SSH. Após a inicialização do experimento, começou-se o processo de configuração dos gateways. Para a implantação, obtivemos os nós de 1 a 120 na

plataforma. Os gateways foram estrategicamente posicionados a cada 25 nós, resultando em três gateways localizados nos nós 25, 50 e 100.

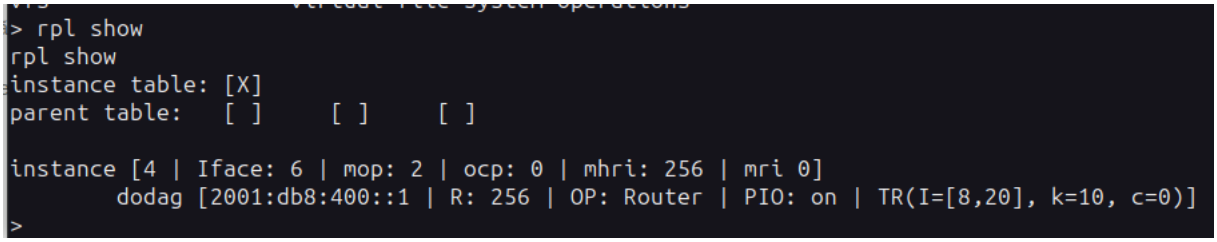
Para os testes, o firmware do gateway foi gravado manualmente em cada um desses três nós por meio da CLI (*Command Line Interface*). Além disso, também foi iniciado um broker MQTT-SN em cada um desses nós e iniciado uma instância da ponte MQTT para a nuvem em cada nó que estava executando um gateway. Após realizar essas configurações na CLI do roteador de borda, podemos analisar a situação inicial das rotas presentes na rede RPL, que continha apenas um dispositivo conectado a rede, conforme apresentado na Figura 19, e a instância RPL, como mostrado na Figura 20.

Figura 19 – Visualização Inicial Rotas RPL



Fonte: Autoria própria (2023).

Figura 20 – Visualização Instância RPL



Fonte: Autoria própria (2023).

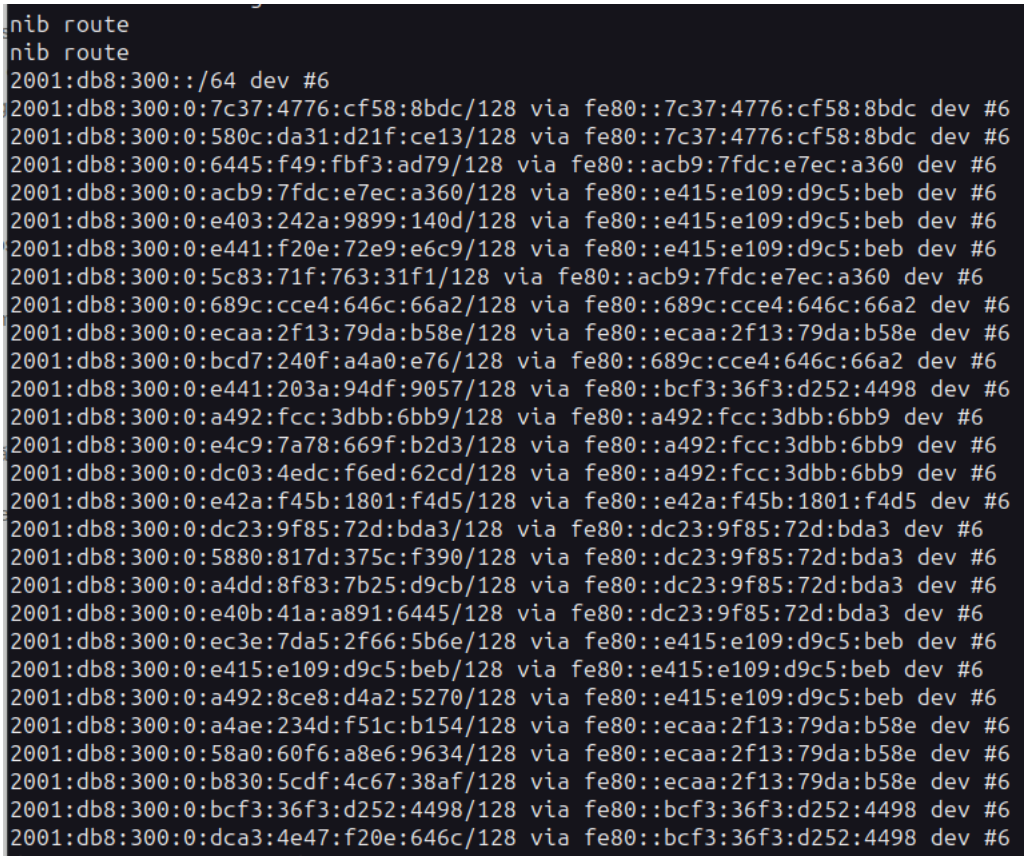
Após essa configuração inicial, um script foi utilizado para automatizar o processo de gravação do firmware do cliente em todos os nós restantes. Após isso toda a rede estava configurada e operando. Para acompanhar os clientes, era possível conectar via interface serial de cada placa e acompanhar os logs do processo, onde se tinha a seguinte visualização conforme a Figura 21.

Após concluir a automação para gravar o firmware nos clientes de todos os nós, executamos o comando para visualizar novamente as rotas no roteador de borda, resultando na visualização apresentada na Figura 22, onde podemos analisar a presença de várias rotas, muitas delas passando por outros dispositivos para chegar até o IP final, característica da rede RPL em árvore.

Figura 21 – Visualização dados via interface serial

Fonte: Autoria própria (2023).

Figura 22 – Rotas após inicialização de todos os clientes



Fonte: Autoria própria (2023).

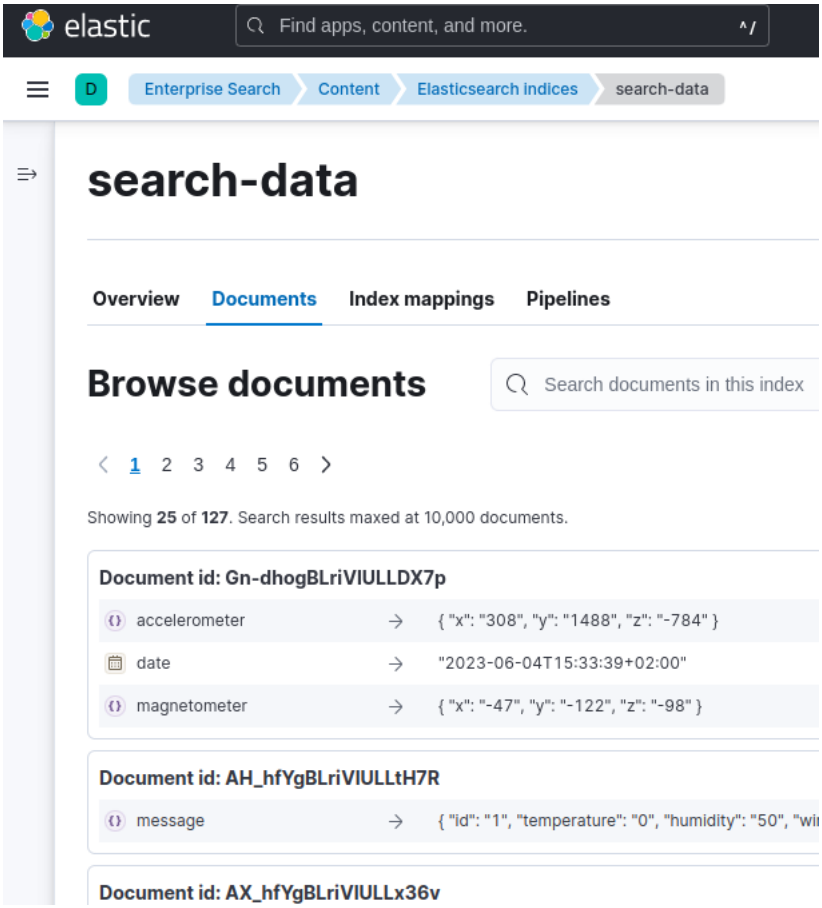
4.2 PONTE ENTRE FIT IOT-LAB E A NUVEM

Uma vez que os clientes estão operacionais e a conexão com o gateway é estabelecida, eles começam a enviar dados periodicamente a cada 10 segundos para um tópico específico. Nesse ponto, o script que atua como uma ponte entra em ação, começando a monitorar as novas mensagens enviadas para o tópico e indexando-as no índice do Elasticsearch.

Este processo é bastante direto. Ele envolve a escuta de um tópico MQTT e, para cada nova mensagem disponível, realiza-se a indexação em um índice do Elasticsearch por meio da

sua SDK (Software Development Kit). Logo após o início do script, os registros começam a ser visualizados na web, conforme mostrado na Figura 23.

Figura 23 – Registros no Elasticsearch



Fonte: Autoria própria (2023).

Ao usar o Elasticsearch para indexar os dados coletados pelos sensores, é possível analisá-los de diversas maneiras. Com a análise das tendências ao longo do tempo, é possível ter uma visão detalhada das mudanças ambientais e identificar rapidamente anomalias, permitindo ações imediatas em caso de leituras anormais. Os painéis de monitoramento em tempo real também permitem acompanhar constantemente os dados dos sensores.

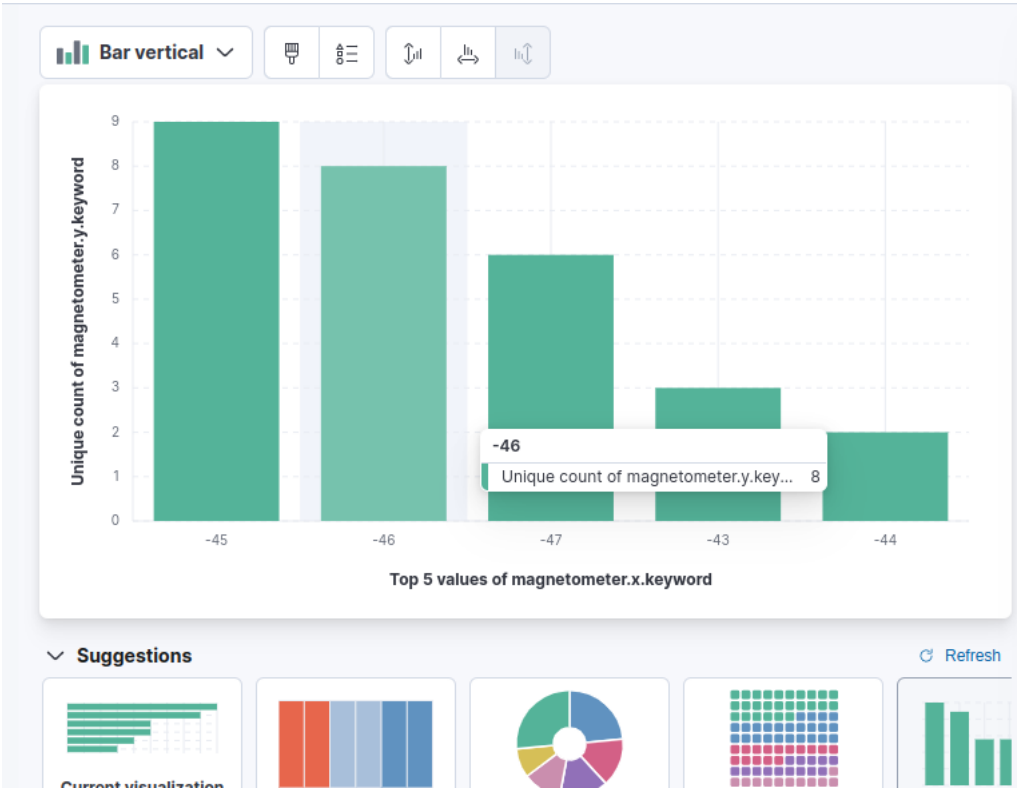
Além disso, a utilização desses dados em modelos de aprendizado de máquina pode antecipar eventos futuros, otimizando a previsão de manutenções e outras necessidades. De maneira geral, os dados estando disponíveis no Elasticsearch, torna-se possível trabalhar em cima desses dados, seja na própria plataforma do elastic cloud ou consumindo os dados diretamente do mecanismo de busca.

4.3 VISUALIZAÇÃO DOS DADOS

Depois que os dados estão disponíveis no Elasticsearch, é possível utilizá-los de várias maneiras para analisar e visualizar informações importantes. Uma das ferramentas mais eficazes para isso é o Kibana. Através dele é possível criar visualizações personalizadas dos dados, com a ajuda de ferramentas como gráficos, tabelas e mapas, podemos apresentar os dados de forma mais visual e fácil de entender.

Após a transferência dos dados dos sensores para a Elastic Cloud, ao criar um novo painel e adicionar uma propriedade, recebemos sugestões de gráficos que podem ser criados. Um exemplo é mostrado na Figura 24, onde é sugerido um gráfico de barras com os valores dos sensores magnéticos das placas A8-M3.

Figura 24 – Dashboards Kibana



Fonte: Autoria própria (2023).

Em resumo, o Kibana oferece várias ferramentas para explorar e analisar dados, permitindo ter uma visão completa e em tempo real dos dados coletados pelos sensores.

4.4 VALIDAÇÃO DA ARQUITETURA

Com a arquitetura toda em execução, foi possível validar o envio de dados coletados pelos sensores das placas A8-M3 passando pela rede RPL por meio do 6LowPan e chegando ao *gateway* no roteador de borda, onde foram posteriormente enviados para a nuvem por meio do *script* de integração, sendo possível visualizar os dados diretamente no Kibana. Nesse processo pode-se validar a eficácia da arquitetura funcionando em seu estado normal, em um cenário sem problemas.

Finalizando os testes com o envio de dados, começou-se os testes de rebalanceamento de carga e tratativa de fluxo em caso de indisponibilidade de um *gateway*. Para validar estes pontos, foi proporcionado uma indisponibilidade em um nó *root* da rede RPL (onde estava rodando o gateway) e analisado o comportamento dos clientes que estavam enviando dados ao mesmo. Foi possível visualizar que instantes após o *gateway* estar indisponível, os clientes tiveram uma reparação global e incorporaram em outra DODAG, conforme a teoria da Figura 10, e consequentemente conseguiram estabelecer conexão com um novo *gateway* e continuar enviando dados, sem ter uma perda significativa de dados.

4.5 DISCUSSÃO DOS RESULTADOS

A arquitetura demonstrou resiliência nos testes de validação realizados. O envio de dados coletados dos sensores a cada dez segundos mostrou-se eficaz para os testes. Como a arquitetura é genérica, o intervalo poderia ser adaptado conforme cada necessidade específica. Após provocar indisponibilidade em um *gateway* que estava operando adequadamente, foi possível validar o balanceamento da rede. Nesse contexto, o resultado observado foi a reconfiguração automática dos nós presentes na árvore, que passaram a buscar outras DODAGs disponíveis na rede. Além disso, foi possível analisar e validar um cenário de saturação, pois ao desligar um *gateway*, houve uma alta demanda de conexão repentina nos outros nós, causando saturação. A implementação e implantação da rede utilizando uma arquitetura física com um volume de placas, e não apenas em um único dispositivo, garantem um cenário mais próximo ao real de uma futura implementação em larga escala dessa arquitetura. Isso proporciona uma previsibilidade muito maior do comportamento em cenários de larga escala. Um dos pontos que poderia ser melhorado para evitar a perda de informações durante processos de troca de *gateway*, quando o nó está em fase de reparação, seria a implementação de um buffer de mensagens para armazenar

as informações durante esse período. Assim, quando o nó estiver conectado novamente, poderá realizar o envio dessas mensagens. Outro ponto que não foi explorado foram os níveis de QoS dentro do MQTT-SN. Trabalhando com o nível 0, que foi utilizado nos experimentos, podem ocorrer algumas perdas de mensagens que poderiam ser tratadas com outros níveis de QoS.

5 CONCLUSÃO

Antes de concluir o trabalho, é pertinente recapitular os objetivos estabelecidos. O objetivo geral consistia em desenvolver uma arquitetura genérica para aplicações IoT, resiliente e tolerante a falhas, com os clientes utilizando o protocolo MQTT-SN. Este amplo objetivo foi desdobrado em metas mais específicas, que englobaram o desenho de uma arquitetura de IoT resiliente e tolerante a falhas, a implantação de uma rede de sensores para avaliar a viabilidade e eficácia da arquitetura proposta, o balanceamento de carga entre os *gateways* para garantir a continuidade do serviço mesmo diante de eventuais indisponibilidades, além da habilidade de enviar dados do *gateway* para a nuvem, possibilitando o processamento e análise desses dados de forma extensa.

Ao longo do desenvolvimento deste trabalho, alcançamos resultados promissores ao implementar uma rede RPL em formato de árvore, que se caracteriza pela capacidade de ser *multi-gateway*, com a presença de mais de um roteador de borda e *gateway*. Para este trabalho, utilizamos placas equipadas com o rádio 802.15.4, no entanto, é importante destacar que a rede RPL é versátil e poderia ser implementada utilizando outros tipos de rádios, como LoRa, Bluetooth ou Zigbee, expandindo assim as possibilidades de aplicação e adaptação em diferentes cenários.

Dada a natureza dos protocolos utilizados e a banda limitada disponível, a rede está sujeita a falhas, portanto, adotar a estratégia de ter caminhos alternativos para a entrega de dados até um *gateway* aumenta significativamente a disponibilidade e confiabilidade dessa arquitetura.

De maneira geral, pode-se afirmar que o trabalho realizado obteve êxito tanto no alcance do objetivo geral quanto dos objetivos específicos. A fase de projeto e implantação foi executada com meticuloso cuidado, e a arquitetura resultante provou ser robusta, resiliente e capaz de atender às demandas de um ambiente IoT atual. Outro aspecto de grande importância para a validação deste trabalho foi a realização de testes em larga escala em uma arquitetura física real, o que permitiu a aproximação a um cenário real.

Quando se trata de arquiteturas em larga escala, o investimento em infraestrutura adicional pode parecer um fator de custo a ser considerado. No entanto, a escalabilidade da rede e o aumento da confiabilidade justificam este investimento. Além disso, em uma arquitetura grande, o esforço extra e os recursos adicionais associados à implementação de múltiplos gateways e roteadores de borda podem ser relativamente pequenos, pois um único nó, como, por exemplo, com uma Raspberry Pi, pode ser suficiente para rodar toda a configuração necessária do roteador

de borda e gateway, tornando os custos adicionais quase insignificantes em comparação aos benefícios adquiridos em termos de resiliência, confiabilidade e desempenho da rede. Essa característica de implementar uma infraestrutura mais robusta sem um impacto significativo no custo é uma grande vantagem, principalmente para organizações que lidam com volumes de dados consideráveis e que precisam garantir a máxima disponibilidade de seus serviços de IoT.

Em relação aos trabalhos futuros, há diversas oportunidades para expandir e aprimorar a arquitetura desenvolvida neste projeto. Uma das possibilidades mais promissoras envolve testar a mesma arquitetura com diferentes protocolos de comunicação, além do 802.15.4, que foi o utilizado neste trabalho. Explorar como a arquitetura se comporta e pode ser otimizada para outros protocolos pode oferecer casos valiosos sobre sua adaptabilidade e versatilidade em variados cenários.

Outro aspecto que poderia ser acrescentado em um futuro trabalho é a possibilidade de trabalhar com diferentes níveis de Qualidade de Serviço (QoS) no MQTT-SN. Neste trabalho, foi utilizado o nível 0, mas explorar outros níveis pode ajudar a entender como variáveis como latência, confiabilidade e eficiência são afetadas nas entregas de mensagens via MQTT-SN nesta arquitetura.

Além disso, a implementação de um buffer de mensagens no cliente RIOT-OS pode ser uma valiosa adição à arquitetura. Isso permitiria que, em cenários onde haja perda de conexão com o gateway, as mensagens possam ser armazenadas localmente. Quando a conexão for reestabelecida, o sistema poderia então enviar as mensagens acumuladas durante o período de indisponibilidade, aumentando significativamente a confiabilidade do sistema. Esse recurso seria particularmente útil em aplicações críticas, onde a perda de dados pode ter consequências significativas.

Em resumo, este trabalho representa uma contribuição valiosa para o campo da IoT, ao introduzir uma arquitetura de alta disponibilidade, versátil, confiável e aplicável a uma ampla gama de cenários e aplicações, reforçando a confiabilidade e resiliência cruciais para os sistemas IoT da atualidade.

REFERÊNCIAS

ASHTON, K. **That Ïnternet of ThingsThing**. [*S. l.: s. n.*], 2010. RFID Journal. Internet of Things. Disponível em: http://www.rfidjournal.com/articles/view?4986. Acesso em: 15 set. 2022.

ASPENCORE; **Integrating IoT and Advanced Technology Designs, Application Development Processing Environments**. [*S. l.: s. n.*], 2019. Embedded Markets Study. Internet of Things. Disponível em: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide. Acesso em: 15 set. 2022.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A Survey. **Computer Networks**, p. 2787–2805, out. 2010. DOI: 10.1016/j.comnet.2010.05.010.

BACCELLI, E. *et al.* RIOT: An open source operating system for low-end embedded devices in the IoT. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 6, p. 4428–4440, 2018.

BAKHOUYA, M. *et al.* Cloud computing, IoT, and big data: Technologies and applications. **Concurrency and Computation: Practice and Experience**, v. 32, n. 17, e5896, 2020. DOI: https://doi.org/10.1002/cpe.5896. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5896. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5896.

BARROS, M. **MQTT Publish/Subscriber - Protocolos para IoT**. [*S. l.: s. n.*], 2015. Embarcados - Sua fonte de informações sobre Sistemas Embarcados. Internet of Things. Disponível em: https://embarcados.com.br/mqtt-protocolos-para-iot. Acesso em: 20 set. 2022.

BORMANN, C.; ERSUE, M.; KERANEN, A. **Terminology for constrained-node networks**. [*S. l.*], 2014.

CAO, K. *et al.* An Overview on Edge Computing Research. **IEEE Access**, v. 8, p. 85714–85728, 2020. DOI: 10.1109/ACCESS.2020.2991734.

COPEL; **Rede Elétrica Inteligente**. [*S. l.: s. n.*], 2020. copel. Rede Elétrica Inteligente. Disponível em: https://www.copel.com/site/copel-distribuicao/rede-eletrica-inteligente/. Acesso em: 12 dez. 2022.

DARABKH, K. A. *et al.* RPL routing protocol over IoT: A comprehensive survey, recent advances, insights, bibliometric analysis, recommendations, and future directions. **Journal of Network and Computer Applications**, v. 207, p. 103476, 2022. ISSN 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103476. Disponível em: https://www.sciencedirect.com/science/article/pii/S1084804522001242.

EINSTEIN, A. O importante é não parar de questionar. A curiosidade tem sua própria razão de existir. **Life**, 1955. Citação atribuída a Albert Einstein publicada no artigo 'Death of a Genius: His fourth dimension, time, overtakes Einstein'8203;“oaicite:"number":1,"metadata":"title":"O importante é não parar de... Albert Einstein -

Pensador","url":"https://www.pensador.com/frase/MTEzNjA2MA/","text":"MILLER, William. Death of a Genius: His fourth dimension, time, overtakes Einstein. Life, 2/5/1955.: Citação atribuída a Albert Einstein publicada no artigo Death of a Genius: His fourth dimension, time, overtakes Einstein, em 2/5/1955, na revista Life","pub$_{d}ate$" : *null*"8203;.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. **International Journal of Information Management**, v. 35, n. 2, p. 137–144, 2015. ISSN 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2014.10.007. Disponível em: https://www.sciencedirect.com/science/article/pii/S0268401214001066.

GUPTA; A. B. E. B. K. B. R. **MQTT Version 5.0**. 02. ed. [*S. l.*], 2018.

HADJ, K. *et al.* **IoT-LAB Infrastructure**. 2015. Disponível em: https://www.researchgate.net/figure/IoT-LAB-Infrastructure_fig1_282667659.

HAMMI, B. *et al.* IoT technologiesfor smart cities. **IET Networks**, v. 7, n. 1, p. 1–13, 2018. DOI: https://doi.org/10.1049/iet-net.2017.0163. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-net.2017.0163. Disponível em: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-net.2017.0163.

HIVEMQ, T. **Quality of Service (QoS) 0,1**. [*S. l.: s. n.*], 2015. hivemq. MQTT Essentials: Part 6. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels. Acesso em: 22 set. 2022.

IDC; **Data Creation and Replication Will Grow at a Faster Rate than Installed Storage Capacity, According to the IDC Global DataSphere and StorageSphere Forecasts**. [*S. l.: s. n.*], 2021. IDC: The premier global market intelligence company. Internet of Things. Disponível em: %3Chttps://www.idc.com/getdoc.jsp?containerId=prUS47560321. Acesso em: 15 set. 2022.

IOT-LAB. **FIT IoT-LAB**. [*S. l.: s. n.*], 2022. FIT IoT-LAB. FIT IoT-LAB. Disponível em: https://www.iot-lab.info. Acesso em: 3 dez. 2022.

JAHNO, P. H. S. **IMPLEMENTAÇÃO DO PROTOCOLO MODBUS UTILIZANDO REDES SEM FIO BASEADO NO PROTOCOLO IEEE 802.15.4**. Nov. 2016. Monografia (TCC) – Universidade Tecnológica Federal do Paraná.

KAUR, N.; SOOD, S. K. Dynamic resource allocation for big data streams based on data characteristics (5Vs). **International Journal of Network Management**, v. 27, n. 4, e1978, 2017. e1978 NEM-16-0189.R2. DOI: https://doi.org/10.1002/nem.1978. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.1978. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1978.

LI, Y. *et al.* Research based on OSI model. In: 2011 IEEE 3rd International Conference on Communication Software and Networks. [*S. l.: s. n.*], 2011. P. 554–557. DOI: 10.1109/ICCSN.2011.6014631.

LOCKE, D. **Mqtt v3. 1 protocol specification. International Business Machines**. 3. ed. [*S. l.*], 2010. p. 42.

MANYIKA, J. **By 2025, Internet of things applications could have $11 trillion impact**. [*S. l.: s. n.*], 2015. McKinsey & Company. Internet of Things. Disponível em: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it. Acesso em: 15 set. 2022.

PAPADOPOULOS, G. Z. *et al.* RFC 4944: Per-Hop Fragmentation and Reassembly Issues. In: 2018 IEEE Conference on Standards for Communications and Networking (CSCN). [*S. l.: s. n.*], 2018. P. 1–6. DOI: 10.1109/CSCN.2018.8581742.

PRADHAN, B. **IoT-Based Applications in Healthcare Devices**. [*S. l.*]: Journal of healthcare engineering, 2010. 2166325998 p. (2021). ISBN 9780451012821.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017.

SILVA JUNIOR, M. P. d. **Análise entre protocolos HTTP e MQTT em projetos IOT**. Nov. 2021. Monografia (TCC) – Universidade Tecnológica Federal do Paraná.

STANFORD-CLARK, A.; TRUONG, H. L. Mqtt for sensor networks (mqtt-sn) protocol specification. **International business machines (IBM) Corporation version**, v. 1, n. 2, p. 1–28, 2013.

TANENBAUM, A. S. **Redes de Computadores: Tradução: Vandenberg D. de Souza**. [*S. l.*]: Rio de Janeiro: Elsevier, 2003.

U-BLOX; **MQTT-SN – lowering the cost of IoT at scale**. [*S. l.: s. n.*], 2020. u-blox. Internet of Things. Disponível em: https://www.u-blox.com/en/blogs/insights/mqtt-sn#:~:text=MQTT%2DSN%20. Acesso em: 20 set. 2022.

VAILSHERY, L. S. **IoT devices installed base worldwide 2015-2025**. [*S. l.: s. n.*], 2016. Statista. Internet of Things. Disponível em: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide. Acesso em: 15 set. 2022.

WINTER, T. *et al.* **IPv6 Routing Protocol for Low-Power and Lossy Networks**. [*S. l.*]: RFC Editor, mar. 2012. 157 p. RFC 6550. (Request for Comments, 6550). Disponível em: https://www.rfc-editor.org/rfc/rfc6550.

YANG, H.; KIM, Y. Design and Implementation of High-Availability Architecture for IoT-Cloud Services. **Sensors**, v. 19, n. 15, 2019. ISSN 1424-8220. DOI: 10.3390/s19153276. Disponível em: https://www.mdpi.com/1424-8220/19/15/3276.