Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Queirós Figueiredo Mburine

Análise de Desempenho dos Protocolos de Aplicação CoAP, MQTT e Radnet Protocol para Internet das Coisas

Rio de Janeiro
2024

Queirós Figueiredo Mburine

Análise de Desempenho dos Protocolos de Aplicação CoAP, MQTT e Radnet Protocol para Internet das Coisas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Profa. Dra. Maria Clicia Stelling de Castro
Coorientador: Prof. Dr. Diego Leonel Cadette Dutra

Rio de Janeiro
2024

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

M478 Mburine, Queirós Figueiredo.
Análise de desempenho dos protocolos de aplicação CoAP, MQTT e Radnet Protocol para internet das coisas/ Queirós Figueiredo Mburine. - 2024.
98 f.: il.

Orientadora: Maria Clicia Stelling de Castro.
Coorientador: Diego Leonel Cadette Dutra
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Internet das coisas - Teses. 2. Internet - Medidas de segurança - Teses. I. Castro, Maria Clicia Stelling de. II. Dutra, Diego Leonel Cadette. III. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. III. Título.

CDU 004.738.5

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta dissertação, desde que citada a fonte.

_____ _____

Assinatura Data

Queirós Figueiredo Mburine

Análise de Desempenho dos Protocolos de Aplicação CoAP, MQTT e Radnet Protocol para Internet das Coisas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 03 de outubro de 2024.
Banca Examinadora:

Profa. Dra. Maria Clicia Stelling de Castro (Orientador)
Instituto de Matemática e Estatística – UERJ

Prof. Dr. Diego Leonel Cadette Dutra (Coorientador)
COPPE – Universidade Federal do Rio de Janeiro

Profa.Dra. Rafaela Correia Brum
Faculdade de Engenharia - UERJ

Profa. Dra. Raquel Coelho Gomes Pinto
Instituto Militar de Engenharia - Exército Brasileiro

Rio de Janeiro
2024

DEDICATÓRIA

Dedico este trabalho à minha família, minha orientadora e meus amigos que foram fundamentais diante de todas as dificuldades no decorrer desta caminhada!

AGRADECIMENTOS

À Deus, que criou o Infinito e Perfeito, permetindo a nossa existência.

Aos meus pais, por sempre terem me apoiado.

Aos Professores Maria Clicia Stelling de Castro e Diego Leonel Cadette Dutra, pela orientação e ajuda empregadas, permitindo a concretização do trabalho.

À Professora Zochil González Arenas pela ajuda proporcionada durante a minha formação.

À minha família, pelo apoio e por terem acreditado em mim.

Aos meus irmãos e aos meus familiares, pelo apoio incondicional.

Ao IME e ao CCOMP, pela oportunidade que me concederam para cursar o mestrado.

Ao ISPG, por ter me autorizado a cursar o mestrado na República Federativa do Brasil.

À CAPES, pelo apoio financeiro.

Muito obrigado a todos que fazem o universo conspirar a nosso favor!

"Tudo que puder ser conectado, será conectado"
(Jacob Morgan)

RESUMO

MBURINE, Queirós Figueiredo *Análise de Desempenho dos Protocolos de Aplicação CoAP, MQTT e Radnet Protocol para Internet das Coisas*. 2024. 98 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

A Internet das Coisas (Internet of Things - IoT) é uma área com avanços significativos, que tem recebido muita atenção dos pesquisadores e da indústria. Ela possibilita a interligação de objetos ou dipositivos de uso diário, hardware programado, software, sensores, atuadores e conexões de rede para atender a uma variedade de aplicações. Vários protocolos da camada de aplicação que permitem esta comunicação já foram estabelecidos. A escolha adequada dos protocolos na camada de aplicação é fundamental para o desempenho, segurança e funcionalidade de uma aplicação específica. Esta camada é a mais próxima do usuário final, desempenhando uma função crucial na interação entre as aplicações e os serviços de rede subjacentes. Assim, esta pesquisa tem como objetivo analisar o desempenho dos protocolos CoAP, MQTT e RadNet em aplicações para agricultura, emulando o ambiente de rede com *Linux network namespaces*. Para os experimentos de todos os protocolos selecionados, foram combinadas três dimensões: três protocolos, quatro métricas quantitativas (RTT, latência, jitter e taxa de entrega de pacotes) e três métricas de escalabilidade (tamanho de mensagens, largura de banda e número de dispositivos). Foram desenvolvidas aplicações para envio e recebimento de mensagens entre clientes e servidores/*brokers*. Os resultados dos experimentos de desempenho são apresentados e discutidos. Estes resultados permitem identificar as diferenças em termos de desempenho entre os protocolos avaliados que precisam ser considerados ao se projetar novas aplicações IoT.

Palavras-chave: Protocolos de Aplicação. CoAP. MQTT. RadNet.

ABSTRACT

MBURINE,Queirós Figueiredo. **Performance Analysis of CoAP, MQTT and Radnet Protocol Application Protocols for the Internet of Things**. 2024. 98f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

The Internet of Things (IoT) is an area with significant advances, receiving much attention from researchers and industry. It enables the interconnection of everyday objects or devices, programmed hardware, software, sensors, actuators, and network connections to meet a variety of applications. Several application layer protocols that enable this communication have already been established. The appropriate choice of protocols in the application layer is fundamental to the performance, security, and functionality of a specific application. This layer is the closest to the end user, playing a crucial role in the interaction between applications and underlying network services. Thus, this research aims to analyze the performance of the CoAP, MQTT, and RadNet protocols in agricultural applications, emulating the network environment with Linux network namespaces. For the tests of all selected protocols, we combine three dimensions: three protocols, four quantitative metrics (RTT, latency, jitter and packet delivery rate), and three scalability metrics (message size, bandwidth, and number of devices). Applications for sending and receiving messages between clients and servers/brokers were developed.

Keywords: Application Protocols. CoAP. MQTT. RadNet.

LISTA DE ILUSTRAÇÕES

Figura 1 - Áreas de Aplicação da IoT. 16
Figura 2 - Requisitos da IoT. 17
Figura 3 - Arquitetura da IoT. 19
Figura 4 - Mensagens do COAP. 22
Figura 5 - Formato de mensagem CoAP. 23
Figura 6 - Arquitetura do MQTT. 24
Figura 7 - Processo publish/subscribe do MQTT. 24
Figura 8 - Formato da mensagem MQTT. 25
Figura 9 - Arquitetura do MQTT-SN. 26
Figura 10 - Elementos de uma RadNet: (a) AP e (b) Cabeçalho de mensagem AP . 27
Figura 11 - Estrutura de Stanza do XMPP. 29
Figura 12 - Arquitetura do XMPP. 30
Figura 13 - Arquitetura do AMQP. 31
Figura 14 - Modelo conceitual do DDS 32
Figura 15 - Configuração de Sistema usando HTTP. 33
Figura 16 - Sequências de Comunicação no HTTP. 33
Figura 17 - Topologia do DODAG. 35
Figura 18 - Pilhas dos Protocolos IP e 6LowPAN 37
Figura 19 - Pilha do Protocolo BLE 38
Figura 20 - Arquitetura do LTE-A 40
Figura 21 - Rede de Namespaces criada. 57

LISTA DE TABELAS

Tabela 1 - Principais características dos protocolos de aplicação. 34
Tabela 2 - Principais características dos protocolos de infraestrura. 42
Tabela 3 - Principais características dos simuladores. 47
Tabela 4 - Tabela comparativa entre trabalhos relacionados e a pesquisa desta dissertação. 52
Tabela 5 - RTT do CoAP 74
Tabela 6 - RTT do MQTT 75
Tabela 7 - Latência do CoAP 77
Tabela 8 - Latência do MQTT 78
Tabela 9 - *Jitter* do CoAP. 79
Tabela 10 - *Jitter* do MQTT 81
Tabela 11 - *Jitter* do RadNet. 82
Tabela 12 - Comparação dos protocolos CoAP e MQTT com base no RTT 83
Tabela 13 - Comparação dos protocolos CoAP, MQTT e RadNet com base no *jitter*. 85

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
RFID	Radio-Frequency Identification
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
IEEE	Institute of Electrical and Electronics Engineer
DSSS	Direct Sequence Spread Spectrum
ISM	Industrial Scientific and Medical
WPAN	Wireless Personal Area Network
SIG	Special Interest Group
QoS	Quality of Service
CoAP	Constrained Application Protocol
REST	REpresentational State Transfer
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
UDP	User Datagram Protocol
MQTT	Message Queue Telemetry Transport
M2M	Machine-to-Machine
TCP	Transmission Control Protocol
IP	Internet Protocol
SMQTT	Secure Message Queue Telemetry Transport
SSL	Secure Sockets Layer
TLS	Transport Layer Security
XMPP	Extensible Messaging and Presence Protocol
XML	Extensible Markup Language
SASL	Simple Authentication and Security Layer
AMQP	Advanced Message Queuing Protocol
DDS	Data Distribution Service
OMG	Object Management Group
MQTT-SN	Message Queue Telemetry Transport for Sensor Networks
RP	RadNet Protocol
RPL	Routing Protocol for Low Power and Lossy Networks
LowPAN	Low Power Wireless Personal Area Networks
BLE	Bluetooth Low Energy
EPCglobal	Electronic Product Code
LTE-A	Long Term Evolution-Advanced
LLNs	Low-power and Lossy Networks

MAC	Medium Access Control
CSMA/CA	Carriers Sense Multiple Access with Collision Avoidance
OFDMA	Orthogonal Frequency Division Multiple Access
MTU	Maximum Transmission Unit

SUMÁRIO

	INTRODUÇÃO	13
1	**CONCEITOS DE INTERNET DAS COISAS**	15
1.1	**Áreas de Aplicação de Internet das Coisas**	15
1.2	**Requisitos da Internet das Coisas**	17
1.3	**Arquitetura para Internet das Coisas**	18
1.4	**Tecnologias de comunicação**	19
1.5	**Protocolos de comunicação**	21
1.5.1	Protocolos de Aplicação	21
1.5.2	Protocolos de Infraestrutura	35
2	**SIMULADORES, *TESTBEDS* E EMULADORES**	43
2.1	**Simuladores**	44
2.2	***Testbeds***	48
2.3	**Emuladores**	49
3	**TRABALHOS RELACIONADOS**	51
4	**AMBIENTE DE EXECUÇÃO**	53
4.1	**Métricas**	53
4.2	**Linux Namespace**	54
4.3	**Network Namespace**	56
4.4	**Cenários dos Experimentos**	57
4.5	**Ambiente dos experimentos**	58
4.6	**Aplicação de IoT no domínio da agricultura**	60
4.7	**Protocolos de comunicação**	62
4.8	**Experimentos Realizados**	72
5	**ANÁLISE DOS RESULTADOS**	73
5.1	**Round Trip Time**	73
5.2	**Latência**	77
5.3	***Jitter***	79
5.4	**Taxa de entrega de pacotes**	83
5.5	**Comparação dos protocolos com base no RTT**	83
5.6	**Comparação dos protocolos com base no *jitter***	84
5.7	**Discussão**	85
	CONSIDERAÇÕES FINAIS	89
	REFERÊNCIAS	92

INTRODUÇÃO

Tem-se observado a inegável popularização da Internet nos últimos anos. Ela está relacionada ao rápido desenvolvimento da indústria de tecnologia, com a evolução de dispositivos de detecção de fenômenos (sensores), computação eficiente, análise de *big data* e comunicação máquina a máquina (M2M). Tem ocorrido um aumento considerável de dipositivos interconectados entre si e conectados à Internet, com o objetivo de trocar informações, para atender diversas necessidades nos mais diferentes domínios.

Gradualmente, a humanidade está cada vez mais dependente da tecnologia e da Internet, em particular, para troca de informações. Nos próximos anos, máquinas inteligentes ou tecnologias de objetos inteligentes serão muito utilizadas (JAIN; CHOUDHARI; SRIVASTAVA, 2021). A Internet das Coisas (*Internet of Things - IoT*) é uma tecnologia que pode ser considerada uma extensão da Internet, permitindo que objetos com capacidade computacional e de comunicação sejam conectados à Internet. A IoT pode conter dispositivos com sensores capazes de captar sons e imagens, realizar cálculos ou se comunicar uns com os outros, além de atuar de forma inteligente para tomada de decisões (SANTOS et al., 2016), (JAIN; CHOUDHARI; SRIVASTAVA, 2021).

A IoT está crescendo em um ritmo muito acelerado, existindo atualmente mais objetos conectados que humanos. Em 2017 existiam 8,4 bilhões de dispositivos conectados, 20,4 bilhões em 2020 e estima-se que em 2030 existirão 500 bilhões de dispositivos IoT. Estes números mostram que a IoT está cada vez mais presente e mudando o cotidiano das pessoas em vários domínios (AHMAD; LAPLANTE; DEFRANCO, 2020).

A IoT possibilita a interligação de objetos ou dipositivos de uso diário, hardware programado, software, sensores e conexões de rede para atender a uma variedade de aplicações. Ela pode ser aplicada em diversas áreas tornando-as inteligentes, como Saúde, Transporte, Agricultura, Cidades, Varejo, entre outras (JAIN; CHOUDHARI; SRIVASTAVA, 2021).

Os dispositivos comunicam-se usando diferentes protocolos de comunicação, através da implementação de suas regras, sintaxe e semântica. Os protocolos de comunicação usados na IoT são vários. Como exemplo podemos citar o CoAP (ALHAIDARI; ALQAHTANI, 2020), MQTT (MQTT Org, 2022b) e RadNet (DUTRA; MORAES; AMORIM, 2012), podendo ser *open source* ou proprietários.

Tecnologias de comunicação como Wi-Fi (DENG et al., 2020), Bluetooth (PADGETTE; SCARFONE; CHEN, 2017), ZigBee (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011), possibilitam que diferentes dispositivos troquem informações entre si, desde que possuam uma interface de comunicação adequada. Por exemplo, um *smart watch* pode se comunicar com um *smartphone* para troca de informações como a distância percorrida, número de passos ou ritmo dos batimentos cardíacos. Estas informações

posteriormente podem ser processadas por uma aplicação de monitoramento de saúde instalado no *smartphone*.

Por ser uma área emergente, a IoT enfrenta vários desafios como heterogeneidade de dispositivos, escalabilidade, interoperabilidade, segurança e privacidade, padronização, eficiência energética entre outros fatores. Ao se projetar uma aplicação IoT é importante considerar estes aspetos para garantir que a futura aplicação atenda as necessidades dos usuários.

Na agricultura, a aplicação da IoT pode ajudar a resolver alguns problemas que o setor enfrenta, como é o caso de esgotamento da água, controle da umidade ou empobrecimento da fertilidade do solo devido às altas taxas de aplicação de insumos.

Como a troca de informações em aplicações IoT é fundamental, o objetivo desta pesquisa consiste em analisar o desempenho dos protocolos CoAP, MQTT e RadNet em aplicações para agricultura emulando o ambiente de rede com *linux network namespaces*.

Esta pesquisa está estruturada da seguinte forma. O Capítulo 1 resume os conceitos básicos sobre IoT que apoiam esta pesquisa, ressaltando as tecnologias de comunicação, protocolos de comunicação e protocolos de infraestrutura. O Capítulo 2 descreve alguns simuladores, *testbeds* e emuladores utilizados no contexto de IoT. O Capítulo 3 descreve os trabalhos relacionados. O Capitulo 4 apresenta o ambiente de execução proposto para a análise de desempenho dos protocolos selecionados. O Capítulo 5 apresenta e discute os resuldados obtidos nos experimentos. A pesquisa é finalizada com apresentação das considerações finais e sugestões para trabalhos futuros, no Capítulo 6.

1 CONCEITOS DE INTERNET DAS COISAS

O termo Internet das Coisas (*Internet of Things - IoT*) pode ser entendido como um ecossistema de computação formado por objetos físicos (coisas) interconectados com a Internet, cujo objetivo é facilitar o cotidiano das pessoas. Quando as coisas (dispositivos) são conectadas à Internet, elas podem compartilhar entre si, processar, armazenar e analisar um enorme volume de dados (MAGRANI, 2021).

A IoT é um paradigma emergente, cuja ideia inicial e o termo surgiram em 1999 com Kevin Aston, quando trabalhou no *Auto-ID Center* do MIT (*Massachusetts Institute of Technology*) em um projeto para desenvolvimento de uma rede de objetos conectados via identificação por rádio frequência (*Radio-frequency Identification - RFID*) (SOBIN, 2020).

Neste capítulo são abordadas as áreas de aplicação, os requisitos de implementação, a arquitetura, as tecnologias de comunicação e os protocolos de comunicação envolvidos em IoT.

1.1 Áreas de Aplicação de Internet das Coisas

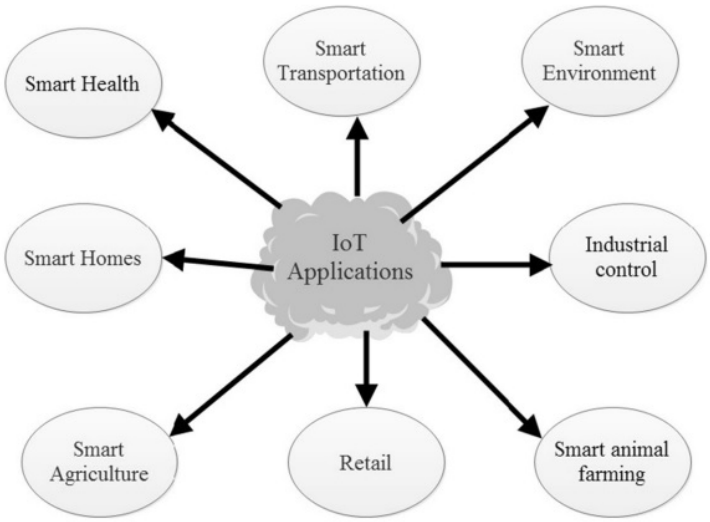
A IoT pode ser aplicada nos mais diversos domínios, como mostra a Figura 1 (SOBIN, 2020). Como exemplo, são destacados e definidos os seguintes domínios: saúde inteligente (*smart health*), casas inteligentes (*smart homes*), agricultura inteligente (*smart agriculture*), transporte inteligente (*smart transportation*), varejo (*retail*), ambiente inteligente (*smart environment*), controle industrial (*industrial control*) e criação inteligente de animais (*smart animal farming*). Cada um destes domínios está descrito a seguir.

1. ***Saúde inteligente:*** consiste em integrar recursos de IoT em dispositivos médicos para prover serviços de qualidade aos pacientes, principalmente aos que necessitam de supervisão constante. Para este domínio de aplicação pode-se destacar dispositivos e/ou aplicações para detecção de quedas em idosos, monitoramento da temperatura em refrigeradores que armazenam medicamentos e vacinas, monitoramento de atividades físicas entre outros.

2. ***Casas inteligentes:*** neste domínio, a IoT torna possível o controle de dispositivos da casa usando um *laptop* ou *smartphone*, por exemplo. Entre as aplicações para casa inteligente pode-se destacar o controle de consumo de energia, temperatura, câmeras de vigilância e aparelhos inteligentes, com o objetivo de proporcionar vida inteligente às pessoas.

3. ***Agricultura inteligente:*** atualmente a agricultura preocupa-se não apenas com a

Figura 1 - Áreas de Aplicação da IoT.



Fonte: (SOBIN, 2020)

produção e produtividade, mas também com a qualidade dos produtos que oferece. Com a aplicação de IoT na agricultura torna-se possível fazer o monitoramento de vários parâmetros importantes para o alcance de bons resultados como o monitoramento da umidade do solo, de condições climáticas, da fertilidade e lactação de gado.

4. ***Transporte inteligente:*** para esse domínio, pode-se recorrer a IoT para monitorar as condições de sistemas de transporte com dispositivos e/ou aplicações para controle de rotas, monitorar as condições de qualidade no embarque de mercadorias, monitoramento de tráfego.

5. ***Varejo:*** no varejo pode-se aplicar a IoT para diversos fins, como por exemplo, rastreamento de produtos, monitoramento das condições de armazenamento de produtos, controle e reposição automática de estoque.

6. ***Ambiente inteligente:*** a área de gestão ambiental preocupa-se com a administração de atividades, tanto econômicas quanto sociais voltadas à utilização racional de recursos naturais, visando à sustentabilidade. Portanto, a aplicação de IoT é importante e necessária na medida em que auxilia na prevenção de desastres naturais, no monitoramento das condições ambientais, na detecção de emissões de CO2 por fábricas e carros entre outros.

7. ***Controle industrial:*** a IoT tem diversas aplicações na indústria podendo se destacar o monitoramento da qualidade do ar, temperatura, presença de ozônio. No que

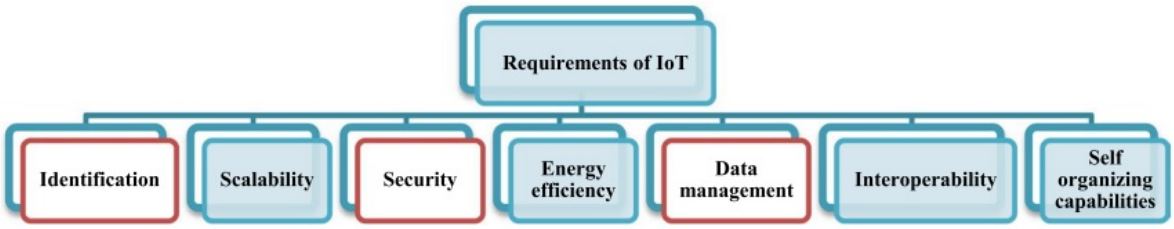
diz respeito à segurança, é possível monitorar o acionamento de alarmes em casos de emergência, entre outras aplicações que a IoT pode proporcionar à indústria.

8. ***Criação de animais inteligente:*** com a aplicação de IoT em fazendas é possível monitorar a movimentação do gado usando sensores que são colocados em animais, monitorar a fertilidade e lactação, garantindo dessa forma maior produtividade.

1.2 Requisitos da Internet das Coisas

Mesmo com a popularização da IoT na atualidade, vários são os desafios que ela enfrenta para uma implementação eficiente como uma tecnologia global. Alguns desses desafios são a falta de protocolos e arquitetura padrão, segurança e privacidade, heterogeneidade de dispositivos, escalabilidade, eficiência energética, interoperabilidade e gerenciamento de dados. A Figura 2 ilustra os principais requisitos da IoT (SOBIN, 2020), que estão abordados a seguir.

Figura 2 - Requisitos da IoT.



Fonte: (SOBIN, 2020)

1. ***Identificação e escalabilidade:*** a IoT interconecta uma quantidade enorme de objetos, por isso, questões de escalabilidade surgem em diferentes níveis, como nomenclatura e endereçamento, comunicação de dados, gerenciamento de informação, provisionamento e gerenciamento de serviços.

2. ***Capacidade de auto-organização:*** na Internet, convencionalmente, os computadores precisam ser configurados pelos usuários. Na IoT, devido às suas exigências, esta configuração precisa ocorrer sem ou com pouca intervenção humana, por isso, os objetos inteligentes precisam se organizar, configurar e adaptar de forma autônoma. As capacidades de auto-organização incluem a descoberta automática de recursos (físicos e de software) e a capacidade de ajustar de forma adaptativa o comportamento de protocolos.

3. ***Interoperabilidade:*** a IoT caracteriza-se por permitir a interconexão de diferentes dispositivos em termos de computação, comunicação, largura de banda, etc. Esses dispositivos são, na maior parte das vezes, de diferentes fabricantes, o que resulta em um alto grau de heterogeneidade de dispositivos e de protocolos de comunicação. Para facilitar a comunicação e cooperação entre diferentes tipos de dispositivos, torna-se necessário o estabelecimento de um padrão comum.

4. ***Gerenciamento de dados:*** ambientes de IoT têm-se caracterizado pela presença de redes de sensores que geram grandes quantidades de dados para a tomada de decisão. Portanto, quando um dispositivo precisar de algum serviço, a informação sobre o servidor que fornece o serviço solicitado deve estar disponível.

5. ***Segurança e privacidade:*** a segurança é um dos requisitos principais em ambientes IoT e deve ser considerada ao se projetar a sua arquitetura, pois ataques à segurança são possíveis, principalmente em ambientes sem fio. A implementação de políticas de segurança em IoT é um desafio devido a fatores como a natureza restritiva dos dispositivos e largura de banda limitada.

6. ***Eficiência energética:*** como os dispositivos IoT são restritos, eles têm a energia da bateria limitada, por isso, torna-se necessária a adoção de mecanismos para minimizar o gasto de energia em computação.

1.3 Arquitetura para Internet das Coisas

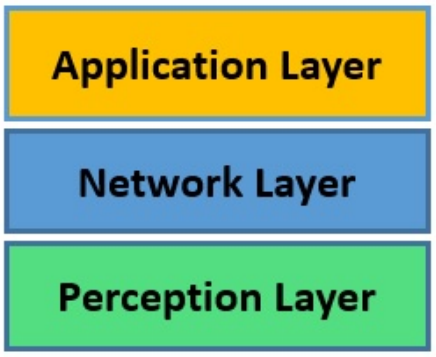
Muitas arquiteturas para IoT já foram propostas em várias pesquisas. Porém, ainda não convergiram para um modelo de referência mesmo existindo alguns projetos com objetivo de trazer uma arquitetura comum que se baseia na análise das necessidades da indústria e dos pesquisadores. Portanto, o modelo básico é o de três camadas nomeadamente: (i) camada de percepção; (ii) camada de rede; e (iii) camada de aplicação, conforme ilustrado na Figura 3 (AL-FUQAHA et al., 2015).

A camada de percepção, que também é conhecida como "camada de dispositivos", "camada sensorial" ou "camada de reconhecimento", inclui as tecnologias usadas para detecção, identificação, atuação e comunicação (GUPTA; QUAMARA, 2020).

A camada de rede, também conhecida como "camada de transmissão" ou "camada de transporte", é uma integração de redes heterogêneas legadas, tecnologias e protocolos, cuja finalidade é transmitir dados coletados pelos nós de percepção para a unidade de processamento da informação. A transmissão dos dados pode ser feita por canais de comunicação com ou sem fio (GUPTA; QUAMARA, 2020).

A camada de aplicação é responsável por gerenciar e fornecer as aplicações com base nas informações coletadas pela camada de percepção e processadas pela unidade de

Figura 3 - Arquitetura da IoT.



Fonte: (AL-FUQAHA et al., 2015)

processamento, fornecer serviços aos usuários finais, através de diversos dispositivos portáteis e equipamentos terminais, como *smartphones* e *notebooks* (GUPTA; QUAMARA, 2020).

1.4 Tecnologias de comunicação

Para que os dispositivos possam se comunicar é necessário que eles tenham alguma interface de comunicação. Eles podem ter interfaces diferentes dependendo da finalidade do dispositivo, do fabricante entre outros aspectos relevantes. Assim, a seguir são apresentadas algumas das principais tecnologias de comunicação usadas atualmente. Elas são o Wi-Fi, ZigBee e Bluetooth.

Wi-Fi

Wi-Fi (*Wireless Fidelity*) (DENG et al., 2020) é uma tecnologia de comunicação sem fio que permite comunicação entre vários dispositivos e equipamentos com o objetivo de troca de informações.

O Wi-Fi é um termo registrado pela Wi-Fi Alliance para designar as WLAN (*Wireless Local Area Network*) e que implementa o padrão IEEE 802.11. Este padrão está em constante evolução, sendo que a versão mais atual é IEEE 802.11ax, lançada em 2021 e que pode atingir até 9,6 GB/s (KHOROV et al., 2018). O Wi-Fi pode operar em duas frequências, de 2,4GHz e 5GHz.

Desde a sua adoção na década de 1990 residências, empresas e *hotspots* estão cada vez mais dependentes de Wi-Fi como rede de acesso. Vários dispositivos Wi-Fi, incluindo computadores pessoais, *smartphones*, televisões, *tablets* ou sensores, usam Wi-Fi como tecnologia de comunicação (DENG et al., 2020).

Em uma rede Wi-Fi a conexão não depende da localização dos dispositivos, pois

ela é estabelecida através de *links* sem fio, em vez de uma infraestrutura fixa cabeada.

ZigBee

O ZigBee (RAMYA; SHANMUGARAJ; PRABAKARAN, 2011) é uma tecnologia de comunicação sem fio desenvolvida pela ZigBee Alliance, que utiliza os serviços de transporte da especificação de rede IEEE 802.15.4.

A energia necessária para o funcionamento de dispositivos ZigBee é menor, e eles podem oferecer um alcance de até 150 metros em ambientes externos, usando a técnica DSSS (*Direct Sequence Spread Spectrum*). Ele opera em diferentes frequências, de acordo com a localização geográfica. Na Europa opera na frequência de 868 MHz, 915 MHz na América do Norte e Austrália e 2,4 GHz disponível globalmente, banda ISM (*Industrial, Scientific and Medical*) com taxas de dados de até 20kbps, 40kbps e 250kbps respectivamente.

Algumas razões para o uso do ZigBee: (i) confiabilidade; (ii) suporte a um número maior de nós; (iii) facilidade de implantação; (iv) segurança; (v) baixo custo; (vi) baixo consumo de energia; e (vii) pouca interferência com outros dispositivos sem fio, pois atua em frequências diferentes.

Bluetooth

O bluetooth (PADGETTE; SCARFONE; CHEN, 2017) é uma tecnologia de rede sem fio de curto alcance, usada principalmente para estabelecer as WPANs (*Wireless Personal Area Networks*). Desenvolvido originalmente pela Ericsson, o bluetooth tem a evolução de suas especificações mantidas e desenvolvidas pelo SIG (*Special Interest Group*) e é padronizado pelo IEEE sob a referência IEEE 802.15.1.

O bluetooth vem integrado em muitos tipos de dispositivos empresariais e de consumo, incluindo telefones celulares, *laptops*, automóveis, dispositivos médicos, impressoras, teclados, *mouses* e fones de ouvido. Eles permitem, dessa forma, a criação de redes *ad hoc* entre usuários para transferência de diversos tipos de informações.

Uma característica importante e interessante do bluetooth é que ele não depende de endereço IP. Sua facilidade de implantação e baixo custo, o tornam uma das melhores opções para comunicação de vários dispositivos IoT. Ele é menos propenso a interferências causadas por outros dispostivos operando na mesma frequência de 2,4 GHz. O seu alcance é de até 10 metros.

Algumas vantagens do bluetooth incluem: (i) baixo consumo de energia; (ii) alto nível de integração; (iii) gerenciamento nativo da voz; e (iv) inclui mecanismos para QoS simples e controle de erros (LABIOD; AFIFI; SANTIS, 2007).

1.5 Protocolos de comunicação

Em um sistema de comunicação, a transmissão de informação entre duas ou mais entidades pode ser viabilizada por um conjunto de regras. Este conjunto de regras denomina-se por protocolos de comunicação. Os protocolos são definidos por regras, sintaxe e semântica, sua implementação pode ser feita pelo hardware, software ou ambos (HOFER-SCHMITZ; STOJANOVIĆ, 2020).

A seguir são apresentados alguns os protocolos de aplicação e protocolos de infraestrutura.

1.5.1 Protocolos de Aplicação

A camada de aplicação faz o gerenciamento de aplicações baseando-se nas informações da camada de percepção e fornece serviços aos usuários finais, através de vários dispositivos e equipamentos. Para que isto seja possível recorre-se aos protocolos projetados para esta camada. Portanto, esta seção apresenta alguns protocolos da camada de aplicação nomeadamente CoAP, MQTT, MQTT-SN, RadNet, XMPP, AMQP, DDS e HTTP. Os protocolos CoAP, MQTT e RadNet são utilizados nos experimentos feitos no contexto desta pesquisa.

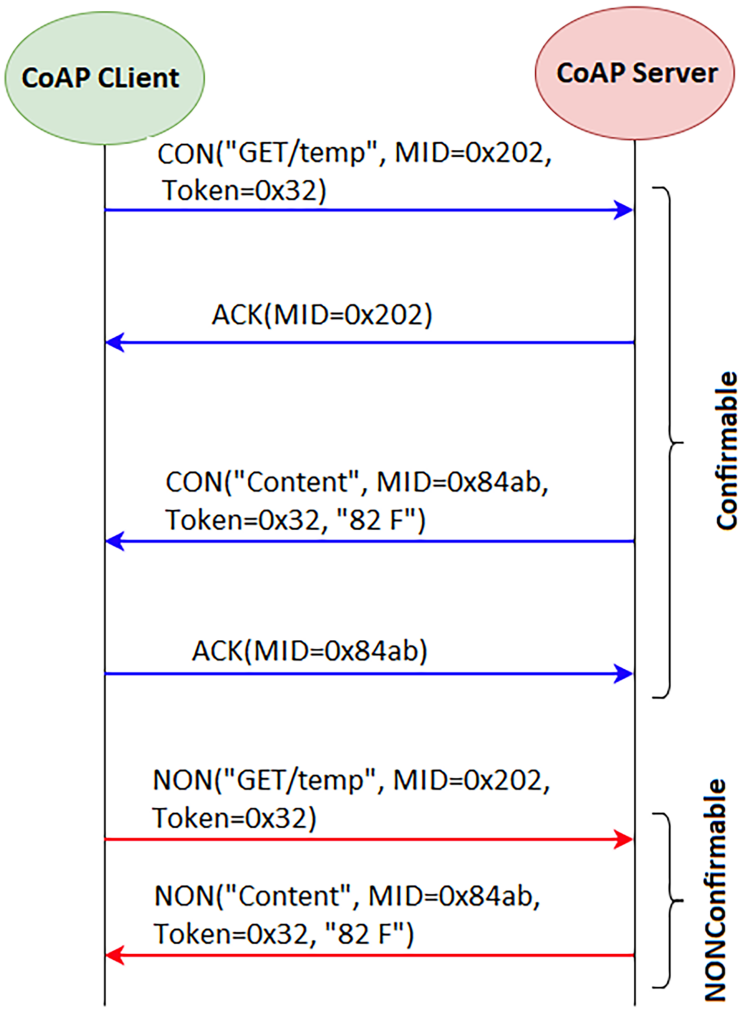
***Constrained Application Protocol* (CoAP)**

O CoAP (ALHAIDARI; ALQAHTANI, 2020) é um protocolo de transferência Web baseado no *REpresentational State Transfer* (REST) e no *Hypertext Transfer Protocol* (HTTP). O REST permite uma troca de dados simples entre clientes e servidores via HTTP. Ele é baseado na arquitetura cliente-servidor e utiliza os métodos `HTTP GET, POST, DELETE e PUT` como verbos e *Uniform Resource Identifiers* (URIs) como substantivos (AL-FUQAHA et al., 2015).

Ao contrário do REST, o CoAP é baseado no *User Datagram Protocol* (UDP) por padrão, tornando-se assim um protocolo mais adequado para aplicações IoT. Devido às restrições de aplicações em IoT, o CoAP modifica algumas funcionalidades do HTTP para atender a requisitos como baixo consumo de energia e operação na presença de *links* não confiáveis, ou seja, onde as desconexões são frequentes (AL-FUQAHA et al., 2015).

Portanto, o CoAP permite que dispositivos com recursos de baixa potência, computação e comunicação utilizem interações RESTful. Ele pode ser dividido em duas subcamadas, a subcamada de mensagens e a subcamada de solicitação e resposta (*request/response*). A subcamada de mensagens detecta duplicações e fornece comunicação confiável sobre a camada de transporte UDP usando um *backoff* exponencial, pois o UDP não fornece mecanismo de recuperação de erro integrado. Por outro lado, a camada de solicitação e resposta lida com as comunicações REST (AL-FUQAHA et al., 2015; STUSEK

Figura 4 - Mensagens do COAP.



Fonte: (KHALIL et al., 2020)

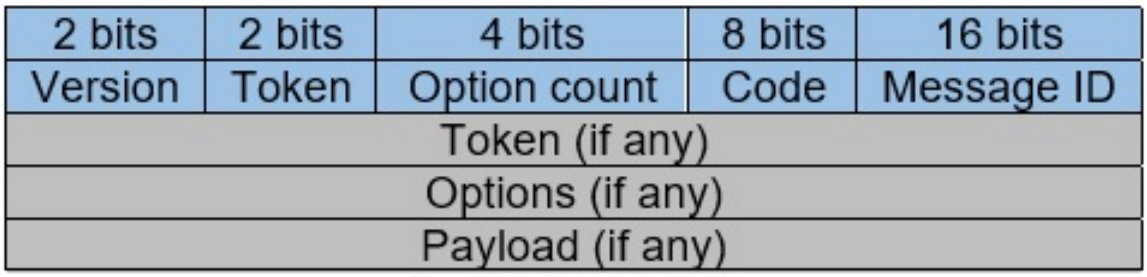
et al., 2019).

Para estabelecer comunicação entre cliente e servidor, o CoAP suporta as mensagens `Conformable` (CON), `Non-Conformable` (NON), `Acknowledgement` (ACK) e `Reset` (RST). A Figura 4 ilustra a comunicação entre cliente e servidor. As mensagens CON carregam consigo solicitação ou resposta e requerem uma mensagem ACK. Por sua vez, mensagens NON são similares a mensagens CON, mas não requerem mensagem ACK. É importante ressaltar que quando o servidor estiver ocupado, mensagens ACK ficam vazias. As mensagens RST são utilizadas quando não se recebe adequadamente mensagens CON ou quando existir algum conteúdo em falta (STUSEK et al., 2019; KHALIL et al., 2020).

À semelhança do HTTP, o CoAP para realizar as operações de `Create, Retrieve, Update e Delete` (CRUD) utiliza os métodos `GET, PUT, POST e DELETE`.

O formato de mensagens CoAP está ilustrado na Figura 5. O cabeçalho é composto por cinco (5) campos de 32 bits no total, nomeadamente: (i) **Version**, que identifica a versão do protocolo CoAP; (ii) **Token**, que especifica o tipo de mensagem, e que pode assumir os valores 0, 1, 2 e 3, para as mensagens `GET, PUT, POST` e `DELETE`, respectivamente; (iii) **Option count**, que indica o comprimento do *token*; (iv) **Code**, que define o método de requisição ou resposta (*request/response*); e (v) **Message ID**, que é usado para identificar mensagens duplicadas e também as respostas correspondentes.

Figura 5 - Formato de mensagem CoAP.



Fonte: Adaptado de (AL-FUQAHA et al., 2015; GUPTA et al., 2021)
.

O CoAP fornece alguns recursos importantes que incluem a observação de recursos, transporte de recursos em blocos, descoberta de recursos, interação usando HTTP e segurança.

No contexto da segurança, o CoAP utiliza o protocolo *Datagram Transport Layer Security* (DTLS). O DTLS é executado sobre a camada UDP e atua como o TLS[1] na camada TCP (AL-FUQAHA et al., 2015), (GUPTA et al., 2021).

***Message Queue Telemetry Transport* (MQTT)**

O MQTT (MQTT Org, 2022b) é um protocolo orientado a mensagens, que destina-se a conectar dispositivos embarcados e redes com aplicações e *middleware*. O MQTT é baseado no mecanismo *publish/subscribe* e considerado ideal para IoT e comunicações M2M. Além disso, o MQTT é capaz de fornecer roteamento para dispositivos pequenos, de baixo custo, de baixa potência e memória em redes não confiáveis ou com baixa largura de banda. Para conexão usa um dos mecanismos de roteamento, um para um, um para muitos, muitos para muitos (AL-FUQAHA et al., 2015; KHALIL et al., 2020).

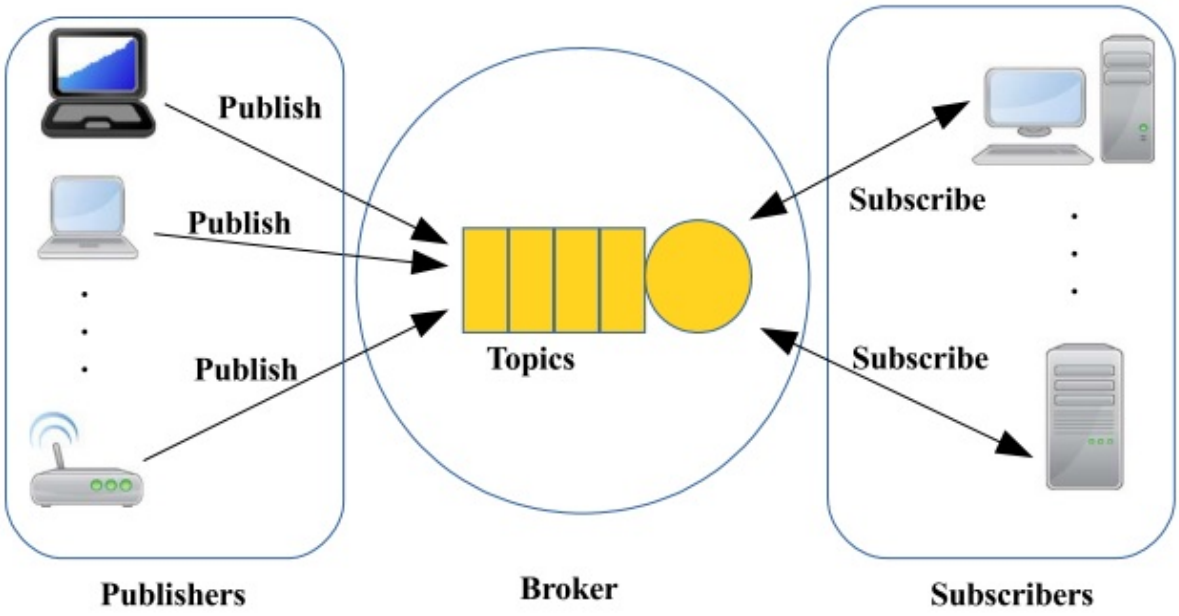
O MQTT é construído sobre o topo do protocolo *Transmission Control Protocol* (TCP). Ele é composto por três (3) componentes nomeadamente: **subscriber, publisher** e **broker**. Dispositivos interessados por um determinado tópico fazem o registro como

[1] O Transport Layer Security é um protocolo de segurança projetado para fornecer segurança nas comunicações sobre uma rede de computadores.

subscribers para posteriormente serem informados pelo **broker** quando os **publishers** publicarem os tópicos de interesse. Depois disso, os **publishers** transmitem as informações aos **subscribers** através do **broker** (AL-FUQAHA et al., 2015; KHALIL et al., 2020).

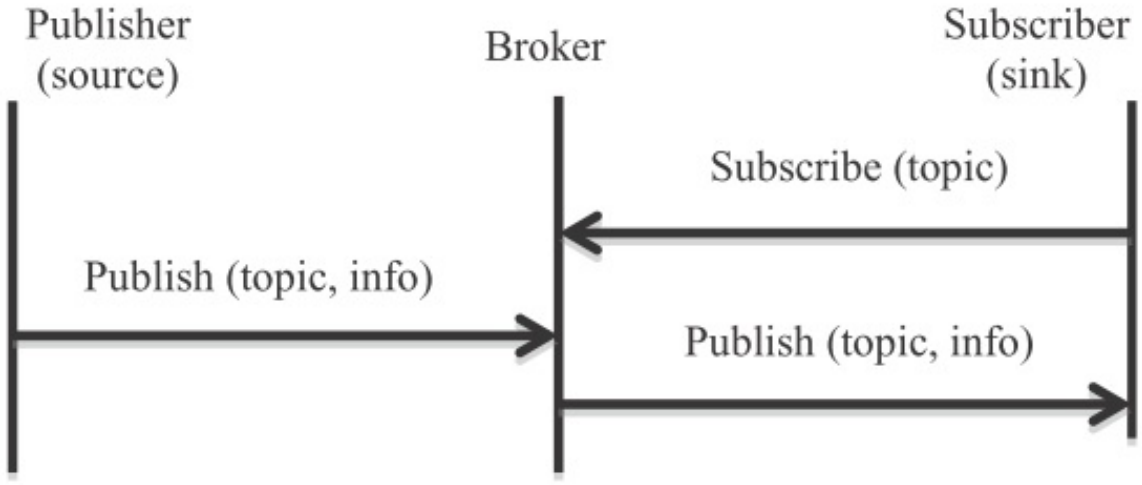
As Figuras 6 e 7 ilustram a arquitetura do protocolo MQTT e o processo **publish/subscribe** do MQTT, respetivamente.

Figura 6 - Arquitetura do MQTT.



Fonte: (AL-FUQAHA et al., 2015)

Figura 7 - Processo publish/subscribe do MQTT.



Fonte: (AL-FUQAHA et al., 2015)

O MQTT pode ser utilizado em diversas aplicações como assistência médica, monitoramento, medidor de energia, transporte, logística e casas inteligentes.

MQTT define o *Quality of Service* (QoS) em três níveis, o que o torna confiável em comparação com outros protocolos. Os três níveis de QoS estão descritos a seguir (GUPTA et al., 2021):

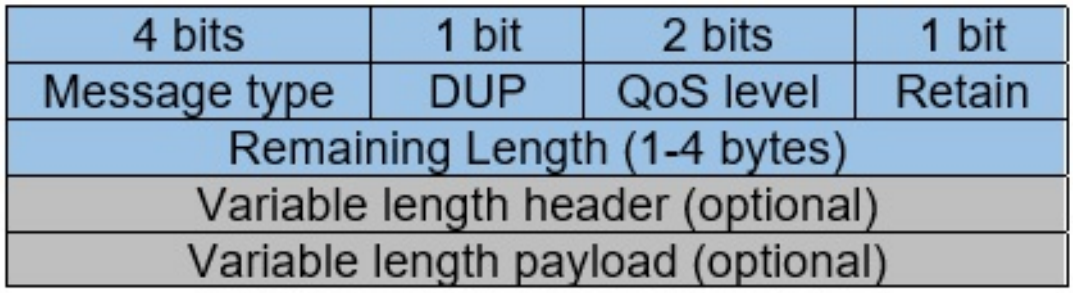
1. ***QoS 0 - Fire and Forget***: é garantido que a mensagem seja enviada corretamente, mas não é esperado o reconhecimento ou confirmação ou recepção da mensagem;

2. ***QoS 1 – Delivered at least once***: garante que a confirmação seja recebida para uma mensagem enviada. Se a confirmação não for recebida dentro do período de tempo definido, o *broker* publica a mensagem novamente;

3. ***QoS 2 – Delivered exactly once***: garante que a mensagem seja entregue apenas uma vez, isto é, sem duplicação.

O formato de mensagens MQTT está ilustrado na Figura 8. O cabeçalho é composto por quatro (4) campos de 8 bits no total, nomeadamente: (i) **Message type**, que especifica o tipo de mensagem incluindo `CONNECT, CONNACK, PUBLISH, SUBSCRIBE` entre outras; (ii) **DUP**, que identifica se a mensagem está duplicada e que o receptor pode tê-la recebido antes, para QoS 1 ou QoS 2; (iii) **QoS level**, que estabelece o nível de QoS 0, 1 ou 2 para a conexão; e (iv) **Retain**, que informa ao servidor para reter a última mensagem de publicação recebida e enviá-la aos novos assinantes como a primeira mensagem.

Figura 8 - Formato da mensagem MQTT.



Fonte: Adaptado de (AL-FUQAHA et al., 2015; GUPTA et al., 2021)

A segurança no MQTT é alcançada na forma de autenticação, onde muitos **publishers** implementam autenticação por meio de um dos pacotes de mensagens `CONNECT`. Os **publishers** e **subscribers** devem validar o nome de usuário e senha corretos antes de estabelecer a comunicação. Um nível de segurança extra, como a criptografia, pode ser implementada como um recurso adicional via *Secure Sockets Layer/Transport Layer Security* (SSL/TLS) (GUPTA et al., 2021).

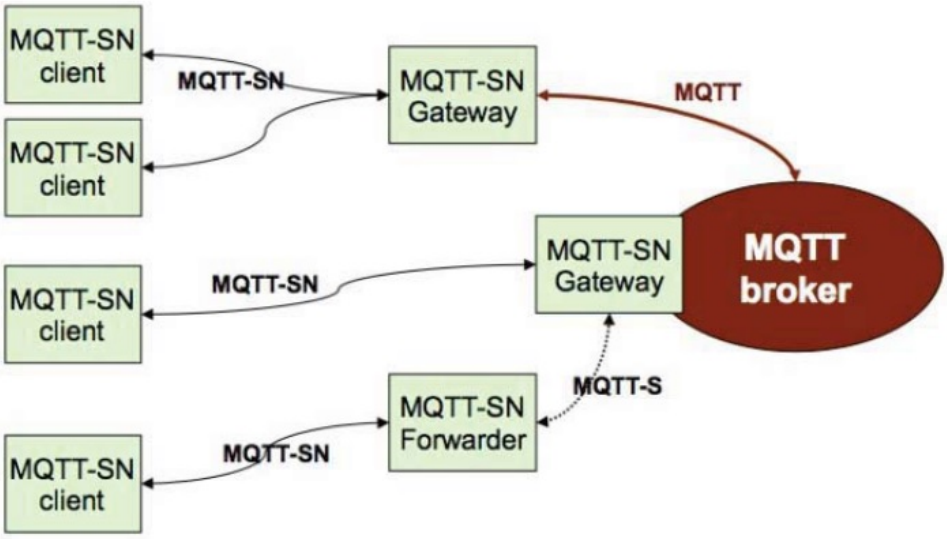
***MQTT for Sensor Networks* (MQTT-SN)**

O MQTT-SN (MQTT Org, 2022a) é um protocolo baseado no MQTT, projetado para ser leve e suportar comunicações sem fio sobre UDP. Portanto, algumas características deste protocolo incluem suporte a falhas de *link*, mensagens de tamanho curto, baixa largura de banda e baixa sobrecarga. O MQTT-SN foi adaptado para lidar com nós restritos tanto de ponto de vista de complexidade, quanto para se adaptar ao fato de que nós restritos não podem ter suporte aos protocolos TCP/IP (SADIO; NGOM; LISHOU, 2019; ROLDÁN-GÓMEZ et al., 2022).

O MQTT-SN difere do MQTT em dois aspetos. Primeiro, ao invés de usar o nome do tópico, usa um *TopicId*, que atua como nome do tópico permitindo a redução do tamanho do pacote. Desta forma, quando um cliente envia uma solicitação de registro ao **broker**, ele indica o ID do tópico com o nome associado para que na próxima solicitação, apenas o ID seja usado. Segundo, o MQTT-SN fornece um mecanismo de *sleeping client* que permite aos clientes se desligarem para economizar energia por um tempo determinado. Ao serem reativados estes clientes podem receber mensagens pelo *buffer* do servidor (SADIO; NGOM; LISHOU, 2019), (ROLDÁN-GÓMEZ et al., 2022).

A arquitetura do MQTT-SN pode ser vista na Figura 9. A função principal dos *gateways* MQTT-SN é a tradução entre MQTT e MQTT-SN. É importante ressaltar que ele usa o MQTT convencional sobre TCP/IP para publicar/subscrever mensagens para/do **broker** (SADIO; NGOM; LISHOU, 2019).

Figura 9 - Arquitetura do MQTT-SN.



Fonte: (SADIO; NGOM; LISHOU, 2019)

Do ponto de vista da segurança, o MQTT-SN não define nenhum mecanismo de segurança no nível do protocolo. No entanto, pode ser usado o Datagram Transport Layer Security (DTLS) para segurança da camada de transporte (SADIO; NGOM; LISHOU, 2019).

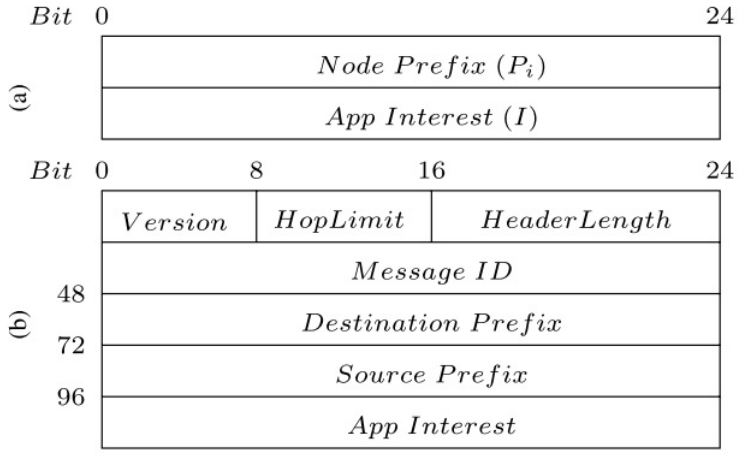
***Protocolo RadNet* (RP)**

O protocolo RadNet (RP) (DUTRA; MORAES; AMORIM, 2012) é uma variação de protocolo para MANETs (*Mobile ad hoc Networks*). O nome RadNets é baseado em (*inteRest-centric mobile ad-hoc networks*). Este protocolo foi desenvolvido no Laboratório de Computação Paralela e Sistemas Móveis (COMPASSO) da Universidade Federal do Rio de Janeiro (UFRJ).

Basicamente, no protocolo RadNet, cada nó usa uma estrutura de dados implementada na camada de rede. Esta estrutura de dados é denominada prefixo ativo (*Active Prefix* - AP). Um AP é composto por um prefixo do nó e pelo interesse da aplicação. O AP possibilita a identificação de nós, endereçamento de mensagens, encaminhamento probabilístico de mensagens, e pesquisa de nomes de forma distribuída. Esse mecanismo possibilita a economia de recursos e reduz a sobrecarga de mensagens (DUTRA; MORAES; AMORIM, 2012).

O protocolo RadNet suporta dois modelos de comunicação: ponto-a-ponto (P2P) e cliente-servidor. Este protocolo pode ser usado em uma variada gama de aplicações, e os dois modelos de comunicação estão descritos em (DUTRA; MORAES; AMORIM, 2012).A Figura 10 ilustra os elementos de uma RadNet. A partir da figura, na parte superior em (a) é possível verificar que um AP é composto pelo campo prefixo do nó (P_i) e pelo campo de interesse da aplicação (I). O campo I é usado por cada aplicação em execução para definir seu interesse atual. Na parte inferior da Figura 10 em (b) podemos ver o cabeçalho de uma mensagem com AP. O cabeçalho contém campos com a versão do RP, o limite de salto, o comprimento do cabeçalho, o identificador de mensagem, dois prefixos de nós para identificar a origem e o destino, e mais um campo com o interesse da aplicação.

Figura 10 - Elementos de uma RadNet: (a) AP e (b) Cabeçalho de mensagem AP



Fonte: (DUTRA; MORAES; AMORIM, 2012)

O uso de interesse permite a uma aplicação em execução em um nó identificar diretamente instâncias da mesma aplicação nos outros nós da rede. O prefixo do nó P_i é

construído como um conjunto de valores probabilísticos, pois ele fornece identificação de nós de forma distribuída com uma probabilidade muito baixa de duplicidade (DUTRA; MORAES; AMORIM, 2012).

Portanto, para fazer o encaminhamento de mensagens o RP utiliza prefixos ativos ao invés de interesses por três razões: (i) evita os problemas associados à nomenclatura que surgem ao usar interesse como uma sequência de caracteres com comprimento variável para um filtro correspondente; (ii) interesses populares podem causar inundações na rede, enquanto interesses impopulares podem impedir que as mensagens cheguem aos seus destinos; e (iii) interesses incompatíveis podem causar o isolamento do nó. O isolamento pode ser minimizado pelo uso de prefixos, pois os campos P_i permitem que os nós vizinhos encaminhem mensagens de forma colaborativa, mesmo que não tenham nenhum interesse comum (DUTRA; MORAES; AMORIM, 2012).

Um AP implementa dois modos de endereçamento nomeadamente: endereçamento origem-destino (*source-to-destination* - S2D) e o endereçamento grupo de interesse (*Interest-Group* - IG), ambos descritos em (DUTRA; MORAES; AMORIM, 2012).

No contexto de segurança, os dados se tornam mais seguros nas mensagens de AP aplicando a criptografia ou utilizando senhas, usando um outro campo de interesse (I).

***Extensible Messaging and Presence Protocol* (XMPP)**

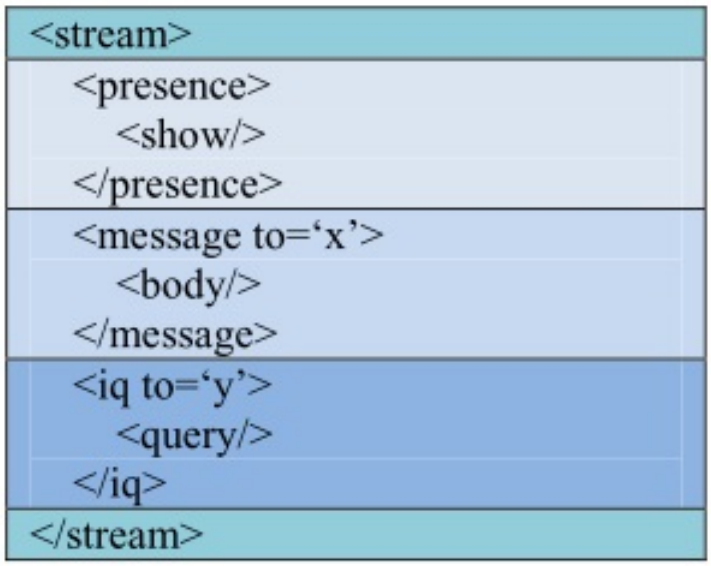
O Extensible Messaging and Presence Protocol (XMPP) (XMPP Org, 2023) é um protocolo de comunicação orientado a mensagens, baseado no *Extensible Markup Language* (XML) e no TCP. Ele permite a troca de pequenas partes de dados estruturados denominados Stanzas XML em tempo real. Assim, o XMPP possibilita uma gama ampla de aplicações como mensagens instantâneas, *chat* privado, chamadas de voz e video, colaboração, conteúdo e roteamento generalizado de dados XML (STANIK; KAO, 2016; NIKOLOV, 2020).

O XMPP pode ser executado em uma variedade de plataformas baseadas na Internet de forma descentralizada, além de ser independente do sistema operacional. Ele possui muitos recursos para aplicações de mensagens instantâneas, que o tornam o protocolo preferido para a maioria de aplicações deste tipo e para outras aplicações de IoT (AL-FUQAHA et al., 2015).

Os *Stanzas XML* especificados pelo XMPP são de três tipos: *message, presence* e *IQ (Info/Query)*. As *stanzas message* são usadas para troca de mensagens entre duas entidades numa rede de sobreposição XMPP. As *stanzas presence* identificam informações de estado do dispositivo com o qual deseja-se comunicar e por fim, *stanzas IQ* permitem o emparelhamento de remetentes e destinatários de mensagens (AL-FUQAHA et al., 2015; STANIK; KAO, 2016; NIKOLOV, 2020; JUN; YANG, 2021). A Figura 11 mostra a estrutura de uma *stanza*.

Todos os *stanzas* têm um tipo de cabeçalho em comum. Os elementos XML raiz $< message/ >$, $< presence/ >$, $e < iq/ >$ devem ter atributos de endereçamento como

Figura 11 - Estrutura de Stanza do XMPP.



Fonte: (AL-FUQAHA et al., 2015)

from e *to*. Adicionalmente, um *stanza IQ* tem dois atributos obrigatórios, o atributo *type* e o atributo *id*, sendo que o *id* associa uma resposta a uma solicitação e o *type* identifica o tipo do *IQ*. Portanto, são especificados quatro tipos: *get, set, result e error*. Os tipos de IQ *get e set* são operações de solicitação e resposta, a resposta pode ser do tipo *result* no caso de processamento da solicitação bem-sucedida. Em caso de erro de processamento da solicitação, é retornado um *stanza* IQ do tipo *error* (STANIK; KAO, 2016).

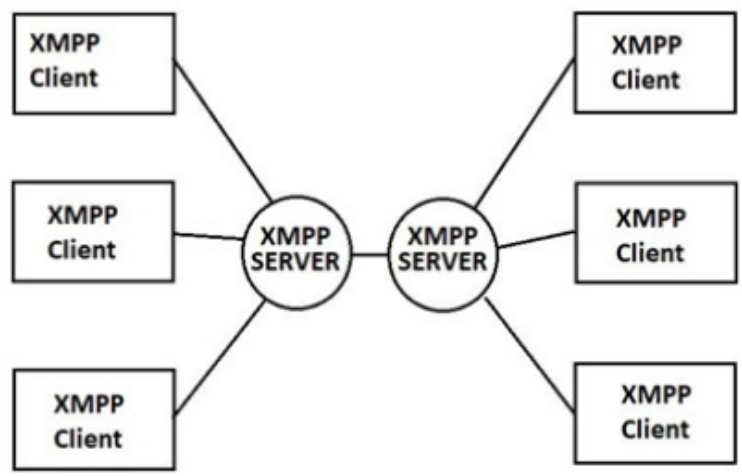
A segurança é um dos principais pontos fortes do XMPP. Ao contrário de outros protocolos, o XMPP possui um recurso de segurança integrado, o TLS, garantindo dessa forma a confidencialidade e integridade dos dados. Nas versões mais recentes do XMPP, outros aspectos de segurança como autenticação, privacidade e controle de acesso são incorporados. Porém, não fornece nenhum nível de qualidade de serviço (QoS) (YUGHA; CHITHRA, 2020), (GUPTA et al., 2021).

A Figura 12 ilustra a arquitetura das comunicações baseadas no protocolo XMPP.

Advanced Message Queuing Protocol (AMQP)

O Advanced Message Queuing Protocol (AMQP) (AMQP Org, 2023) é um protocolo orientado a mensagens, assíncrono e de padrão aberto (*open source*). Ele suporta a comunicação confiável por meio de primitivas de garantia de entrega de mensagens, que incluem entrega no máximo uma vez, pelo menos uma vez e exatamente uma vez (*at-most-once, at-least-once e exactly once*). O principal objetivo do uso do AMQP é permitir que diferentes aplicações e sistemas possam trabalhar em conjunto de modo a melhorar a interoperabilidade (AL-FUQAHA et al., 2015; PATHAKA; TEMBHURNE, 2018).

Figura 12 - Arquitetura do XMPP.



Fonte: (AL-FUQAHA et al., 2015)

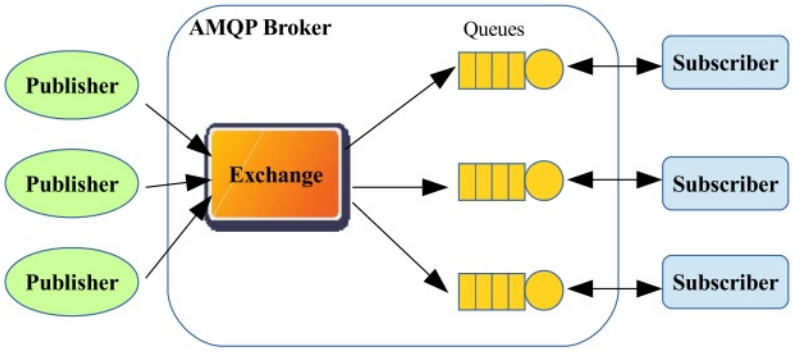
O AMQP oferece suporte para que diferentes aplicações possam enviar e receber mensagens, independentemente das plataformas e linguagens de programação utilizadas. Ele suporta diferentes protocolos de transporte, porém, assume um protocolo de transporte confiável, como o TCP (CHAUDHARY; PEDDOJU; KADARLA, 2017).

As comunicações no AMQP são tratadas por dois componentes principais: *exchanges* e filas de mensagens. Um *exchange* tem a função de rotear as mensagens para as filas apropriadas. As mensagens podem ser armazenadas em filas de mensagens e depois enviadas aos destinatários. As mensagens no AMQP podem ser simples ou anotadas. As mensagens simples são fornecidas pelo remetente e mensagens anotadas são vistas no receptor (AL-FUQAHA et al., 2015).

O AMQP suporta o modelo de comunicação *publish/subscribe* e sua arquitetura é composta por *producer/publisher* que consiste numa aplicação que gera mensagens AMQP, *consumer/subscriber* que é uma aplicação cliente que recebe mensagens AMQP geradas pelo *publisher*. O *broker* é um servidor onde aplicações clientes se conectam para enviar e receber mensagens e que representa abstração de uma fila utilizada para armazenar e enviar mensagens. O *exchange* é responsável por redirecionar e filtrar mensagens enviadas pelos *publishers* (CAIZA et al., 2019).

Quanto à segurança, o AMQP fornece autenticação de entrega com Simple Authentication and Security Layer (SASL) e comunicação segura usando o TLS (MCATEER et al., 2017). A Figura 13 ilustra a arquitetura do AMQP.

Figura 13 - Arquitetura do AMQP.



Fonte: (AL-FUQAHA et al., 2015)

***Data Distribution Service* (DDS)**

O Data Distribution Service (DDS) (DDS Foundtion Org, 2023) é um protocolo desenvolvido pelo Object Management Group (OMG), baseado no TCP e no mecanismo *publish/subscribe* descentralizado para comunicações M2M em tempo real. Ele foi desenvolvido como um *middleware* de rede para contornar as desvantagens da arquitetura *publish/subscribe* centralizada. Portanto, ao contrário de outros protocolos que apresentam na sua arquitetura um *broker*, o DDS é um protocolo *broker-less* que apresenta nós descentralizados de clientes em um sistema que permite que esses mesmos nós se identifiquem como *subscribers* ou *publishers* considerando como base a localização do servidor (AL-FUQAHA et al., 2015; CHEN; KUNZ, 2016).

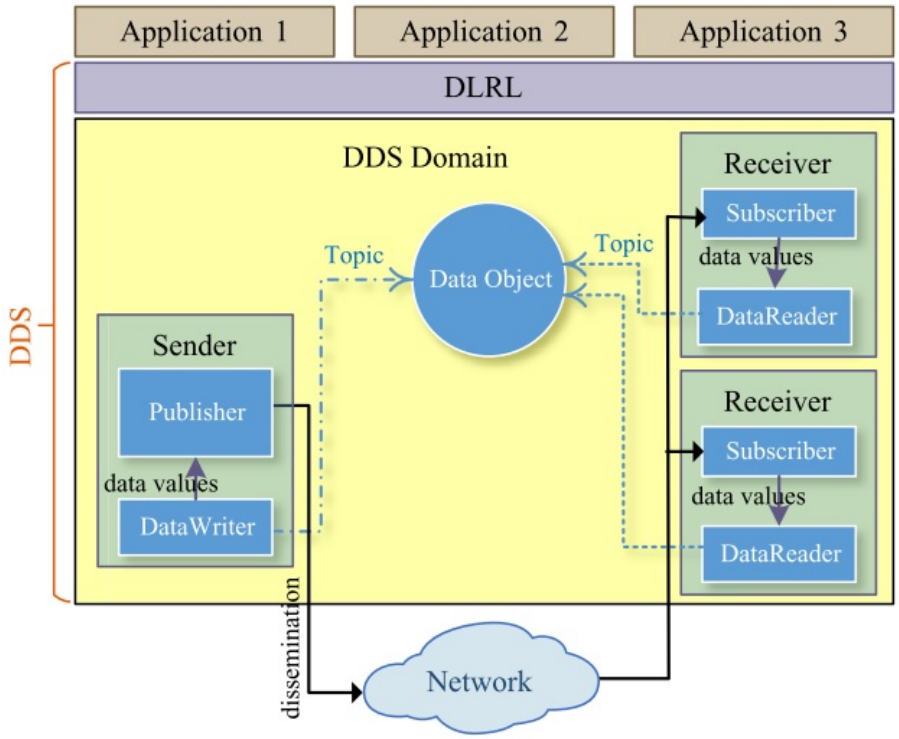
A arquitetura DDS elimina a necessidade dos usuários identificarem onde estão outros nós ou quais são os tópicos de seu interesse. Os nós DDS se autodescobrem na rede e podem enviar e receber dados de forma anônima, baseando-se apenas nos tópicos. Assim que *subscribers* e *publishers* são vinculados, a conexão entre eles passa a ser *peer-to-peer* e o servidor é ignorado (CHEN; KUNZ, 2016).

Em geral, o DDS é usado para *multicasting* com alta confiabilidade para dispositivos IoT de baixa latência. Este protocolo pode ser utilizado em diversos tipos de aplicações, como para gerenciamento de redes inteligentes, aplicações médicas, robótica e simulação (YUGHA; CHITHRA, 2020).

A arquitetura do DDS define duas camadas: *Data-Centric Publish/Subscriber* (DCPS) e *Data-Local Reconstruction Layer* (DLRL). A primeira camada DCPS é responsável pela entrega de informações aos *subscribers*. A segunda camada DLRL é opcional e tem como função realizar a interface para as funcionalidades do DCPS (AL-FUQAHA et al., 2015). Cinco elementos estão envolvidos no fluxo de dados do DCPS: *publisher, dataWriter, subscriber, dataReader* e *topic* (AL-FUQAHA et al., 2015). A Figura 14 ilustra

o modelo conceitual do DDS.

Figura 14 - Modelo conceitual do DDS



Fonte: (AL-FUQAHA et al., 2015)

No conexto da segurança o DDS inclui mecanismos que fornecem autenticação, controle de acesso, confidencialidade e integridade à distribuição de informações. O *DDS Security* usa uma arquitetura *peer-to-peer* descentralizada que fornece segurança sem sacrificar o desempenho em tempo real.

***Hypertext Transfer Protocol* (HTTP)**

O Hypertext Transfer Protocol (HTTP) (Mozilla Org, 2023) é um protocolo base para a arquitetura cliente-servidor usada da Web. Ele define operações como `GET`, `POST`, `PUT` e `DELETE`. O HTTP é executado sobre o TCP/IP garantindo assim uma comunicação confiável (NIKOLOV, 2020).

O HTTP usa Uniform Resource Locator (URI) em vez de tópicos. Durante a comunicação entre cliente e servidor, o cliente solicita um recurso na Internet, o HTTP envia um pacote de dados com cabeçalhos a um URI. Por sua vez, o servidor envia dados através do URI, e o cliente recebe dados através de um URI particular (NAIK, 2017).

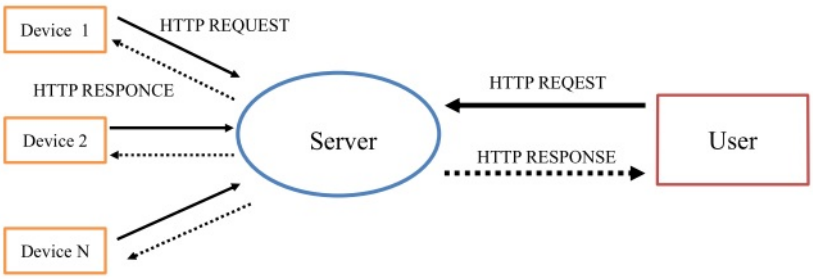
O protocolo HTTP pode fazer transferência de um grande volume de dados em pacotes pequenos. Este fato pode sobrecarregar a rede, causando consequentemente, problemas de largura de banda. Uma solicitação HTTP usa um formato de cabeçalho TCP complexo com nove pacotes que na maioria dos casos são desnecessários, causando desta forma, um desperdício desnecessário de recursos (WUKKADADA et al., 2018).

Apesar da ineficiência, o protocolo HTTP é muito usado em aplicações IoT através do REST, porque o REST fornece uma maneira simples de expor funcionalidades (CRUZ et al., 2018).

Do ponto de vista de segurança, o HTTP fornece comunicação segura usando TLS/SSL (NAIK, 2017).

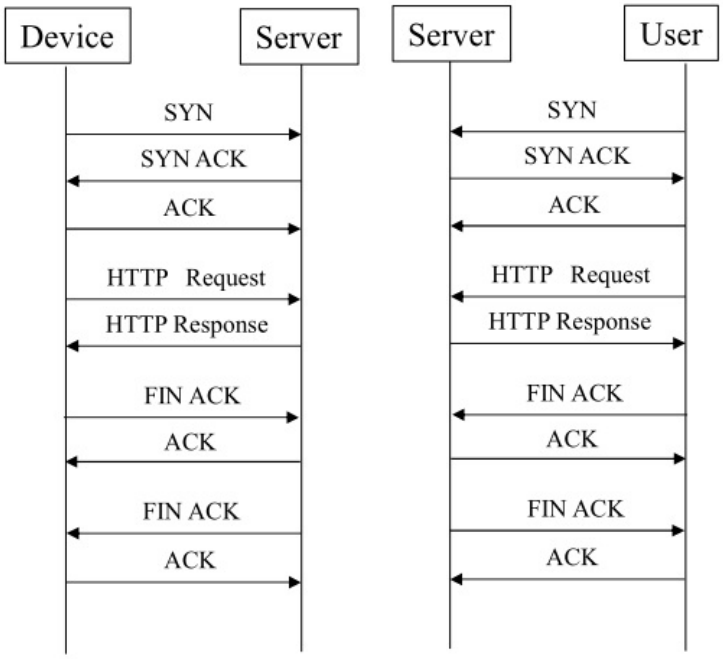
As Figuras 15 e 16 ilustram a configuração de sistema usando HTTP e sequências de comunicação no HTTP, respetivamente.

Figura 15 - Configuração de Sistema usando HTTP.



Fonte: (YOKOTANI; SASAKI, 2016)

Figura 16 - Sequências de Comunicação no HTTP.



Fonte: (YOKOTANI; SASAKI, 2016)

A Tabela 1 resume as principais principais características dos protocolos de aplicação.

Tabela 1 - Principais características dos protocolos de aplicação.

Protocolos de Aplicação	Características	RESTfull	Transporte	Abstração	Segurança	QoS	Cabeçalho (Byte)	Arquitetura C/S
CoAP	Foco em dispositivos com recursos restritos	Sim	UDP	Publicação/Subscrição Requisição/Resposta	DTLS	Sim	4	Sim
MQTT	Flexivel, simples e confiável na entrega de mensagens	Não	TCP	Publicação/Subscrição	SSL/TSL	Sim	2	Não
MQTT-SN	Adequado a redes sem fio com altas taxas de falha na entrega, pouca largura de banda e necessidade de mensagens pequenas	Não	UDP/TCP	Publicação/Subscrição	DTLS	Sim	2	Não
XMPP	Usado para transferência de mensagens instantâneas	Não	TCP	Publicação/Subscrição Requisição/Resposta	SASL/TLS	Não	Variável (XML tag)	Sim
AMQP	Fornece recursos diferentes que incluem o roteamento e armazenamento de mensagens no broker usando filas de mensagens	Não	TCP	Publicação/Subscrição	SASL/TLS	Sim	8	Não
DDS	Escalabilidade, confiabilidade, comunicação em tempo real	Não	UDP	Publicação/Subscrição	DTLS	Sim	4-56	Não
HTTP	Muito difundido e possui recursos úteis, porém, consomem maior energia e largura de banda	Sim	TCP	Requisição/Resposta	SSL/TLS	Não	-	Sim
RP	Foco em redes sem fio centradas em interesse	Não	TCP	Publicação/Subscrição	-	Não	6	Não

Legenda: C/S - cliente/Servidor

Fonte: Adaptado de (AL-FUQAHA et al., 2015; ESFAHANI et al., 2017; NAIK, 2017; GHOTBOU; KHANSARI, 2021; QUINCOZES; QUINCOZES; KAZIENKO, 2021; JUN; YANG, 2021)

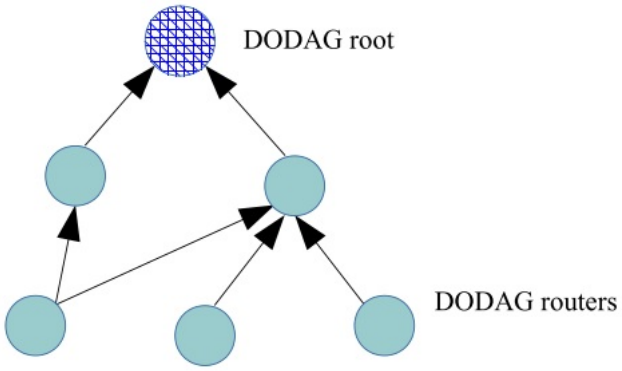
1.5.2 Protocolos de Infraestrutura

Esta seção aborda os seguintes protocolos de infraestrutura: *Routing Protocol for Low Power and Lossy Networks* (RPL), *Low Power Wireless Personal Area Networks* (6LowPAN), IEEE 802.15.4, *Bluetooth Low Energy* (BLE), *Electronic Product Code* (EPC-global), *Long Term Evolution-Advanced* (LTE-A) e *Z-Wave*. Cada um deles está descrito a seguir.

Routing Protocol for Low Power and Lossy Networks (RPL)

O Routing Protocol for Low Power and Lossy Networks (RPL) (SOBRAL et al., 2019) é um protocolo padrão para redes de baixa potência e com perdas (*Low-power and Lossy Networks LLNs*) que organiza a rede física em forma de um grafo acíclico direcionado (*Directed Acyclic Graph - DAG*). Cada DAG nesta topologia está enraizado em um único destino chamado de DAG do destino orientado (*Destination-Oriented DAG - DODAG*). O DODAG representa uma árvore de roteamento criado pelo nó raiz (AL-FUQAHA et al., 2015; SOBRAL et al., 2019; GHALEB et al., 2018). A Figura 17 mostra a topologia do DODAG.

Figura 17 - Topologia do DODAG.



Fonte: (AL-FUQAHA et al., 2015)

O RPL cria o DODAG enviando mensagens `DIO (DODAG Information Object)` para nós vizinhos, contendo o *rank* da rede, modo de operação, funções objetivo (*Objective Functions - OF*) e métricas, que são utilizados para tomar decisões como, por exemplo, se o nó vai ou não aderir ao DODAG (SOBRAL et al., 2019).

Para manter a informação da topologia atualizada, o RPL usa quatro tipos de mensagens: `DODAG Information Object (DIO)`, `Destination Advertisement Object (DAO)`, `DODAG Information Solicitation (DIS)` e `DAO Acknowledgment (DAO-ACk)` (AL-FUQAHA et al., 2015; GHALEB et al., 2018; SOBRAL et al., 2019).

As mensagens `DIO` são usadas na manutenção do DODAG através de envio de informações e parâmetros de configuração, que o novo nó a ser incorporado usa para selecionar

seu pai preferencial em um conjunto de pais candidatos. Para manter a estabilidade da rede, as mensagens `DIO` são enviadas periodicamente pelos nós, com base no *Trickle Timer* (AL-FUQAHA et al., 2015; GHALEB et al., 2018; SOBRAL et al., 2019).

As mensagens `DAO` são usadas pelo RPL para criação de rotas ascendentes (*upward routes*) e descendentes (*downward routes*), com base em informações de destino propagadas pelos nós (GHALEB et al., 2018; SOBRAL et al., 2019).

As mensagens `DIS` são utilizadas por um nó para selecionar um DODAG preferencial, ao qual se conecta, através da solicitação de mensagem `DIO` a um nó que já faz parte do DODAG. Vale ressaltar que a escolha do DODAG é baseada nas funções objetivo (SOBRAL et al., 2019).

Mensagens `DAO-Ack` correspondem a respostas de confirmação à mensagens `DAO`, enviadas pelos receptores destas mensagens `DAO`. Receptores podem ser `DAO` pai ou DODAG raiz (AL-FUQAHA et al., 2015; GHALEB et al., 2018; SOBRAL et al., 2019).

Segundo (SOBRAL et al., 2019), o processo de construção do DODAG caracteriza-se por ser dependente das funções objetivo. O desempenho da rede está diretamente relacionado à seleção de rotas, sendo possível o aumento ou redução do desempenho.

Low Power Wireless Personal Area Networks (6LowPAN)

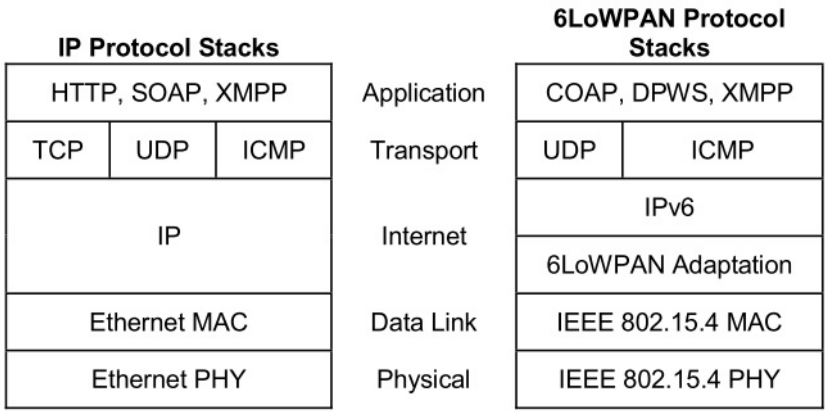
O Low Power Wireless Personal Area Networks (6LowPAN) (OOKO et al., 2020) é uma especificação que permite que dispositivos com recursos limitados e heterogêneos sejam conectados à Internet (THUNGON et al., 2021). O 6LowPAN suporta diferentes tipos de topologias como malha e estrela. Ele propõe uma camada entre a camada MAC (*Medium Access Control*) e a camada de rede para lidar com interoperabilidade entre IEEE 802.15.4 e o protocolo IPv6 (AL-SARAWI et al., 2017). Esta camada também permite a compressão de cabeçalhos, fragmentação de pacotes, encaminhamento de datagramas Ipv6 na camada de *link* de dados, remontagem e encaminhamento de rotas em malha (OOKO et al., 2020).

As redes 6LowPAN se comunicam com o protocolo IP e recorrem aos roteadores de borda para se conectar com outras redes que usam um meio diferente para encaminhamento de datagramas. Os roteadores de borda aprimoram a interoperabilidade fornecendo mecanismos de transição para conexão com redes Ipv4, sem necessidade de implementação do Ipv4 pelos nós 6LowPAN (OOKO et al., 2020).

A Figura 18 mostra a pilha de protocolos do 6LowPAN. Uma pilha do protocolo IP é adaptada com a introdução da camada LowPAN. A camada de rede usa o protocolo Ipv6, permitindo ao 6LowPAN dar suporte a vários dispositivos heterogêneos. A camada física (PHY) permite a comunicação dispositivos através de canais sem fio. A camada MAC controla o acesso ao canal de comunicação usando mecanismos CSMA/CA (*Carriers Sense Multiple Access with Collision Avoidance*) (OOKO et al., 2020).

O 6LowPAN pode ser usado em redes heterogêneas de baixa potência, ou ainda quando há necessidade de comunicação entre diferentes dispositivos, de forma escalável e

Figura 18 - Pilhas dos Protocolos IP e 6LowPAN



Fonte: (OOKO et al., 2020)

com mobilidade. Portanto, ele é viável em aplicações de automação de saúde, automação residencial, gestão e eficiência energética, redes inteligentes, segurança de sistemas entre outras (OOKO et al., 2020).

Uma comunicação segura pode ser alcançada encriptando mensagens entre os dispositivos que se comunicam e a chave entre os dispositivos deve ser acordada para transmissão segura de mensagens (THUNGON et al., 2021).

Bluetooth Low Energy (BLE)

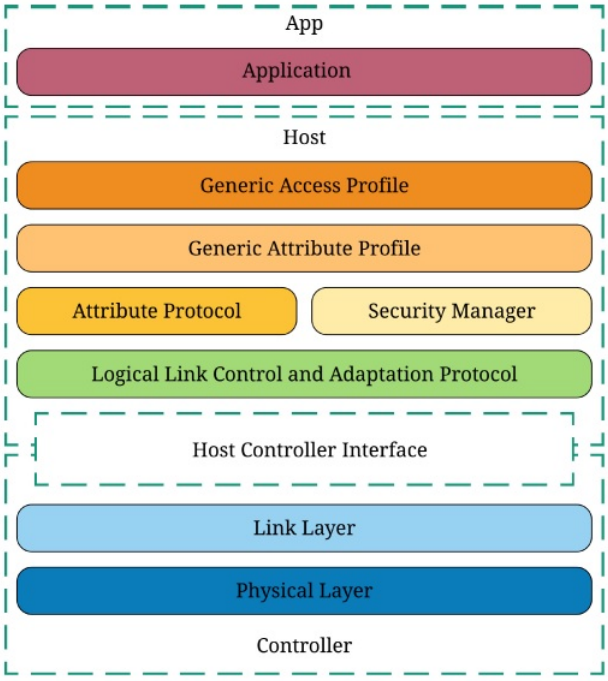
O BLE (BARUA et al., 2022) é um protocolo de comunicação sem fio de baixa potência e de curto alcance para aplicações IoT, que está incluído na maioria dos dispositivos disponíveis atualmente no mercado (GARCIA-ESPINOSA et al., 2018). Segundo (BARUA et al., 2022), o BLE tornou-se num protocolo para IoT devido ao baixo consumo de energia, facilidade de desenvolvimento, boa cobertura da rede e velocidade de transferência de dados.

O uso generalizado do Bluetooth clássico em muitos dispositivos contribuiu significativamente para a adaptação do BLE. As suas implementações são semelhantes e têm suporte de quase todos sistemas operacionais modernos (BARUA et al., 2022).

A pilha do BLE é composta por três camadas principais: camada controladora, camada de *host* e camada de aplicação (BARUA et al., 2022). A Figura 19 ilustra a pilha do protocolo BLE.

A camada controladora inclui hardware para enviar e receber dados, e é composta pela camada física e camada de *link* (BARUA et al., 2022). A camada de *host* permite às aplicações fazer *scan*, descobrir dispositivos, conectar e trocar informações com dispositivos pares. Essa camada é composta pelo Protocolo de Adaptação e Controle de Enlace Lógico (*Logical Link Control and Adaptation Protocol – L2CAP*), Protocolo de Controle de Atributos (*Attribute Control Protocol - ATT*), Gestor de Segurança (*Security Manager - SM*), Perfil de Atributo Genérico (*Generic Attribute Profile - GATT*) e Perfil de Acesso

Figura 19 - Pilha do Protocolo BLE



Fonte: (BARUA et al., 2022)

Genérico (*Generic Access Profile - GAP*) (BARUA et al., 2022). Finalmente, a camada de aplicação, contém a interface da aplicação, sua lógica e estrutura nos dispositivos BLE. Ela tem uma comunicação direta com a camada *host* (BARUA et al., 2022). Entre a camada controladora e a camada *host* encontra-se a interface controladora do *host* (*Host Controller Interface - HCI*) cuja função é facilitar comunicações bidirecionais entre o *host* e o controlador (BARUA et al., 2022).

IEEE 802.15.4

O IEEE 802.15.4 (AL-FUQAHA et al., 2015) é um protocolo para aplicações com baixo consumo de energia, baixa taxa de dados, que fornece comunicação confiável, podendo operar com vários nós em diferentes plataformas (AL-FUQAHA et al., 2015). Portanto, o IEEE 802.15.4 pode ser usado em vários domínios como no controle industrial, automação residencial, cuidados de saúde e redes inteligentes (PARK; FISCHIONE; JOHANSSON, 2013).

Esse protocolo especifica uma sub-camada de controle de acesso ao meio (MAC) e uma camada física (PHY) para redes de área privada de baixa taxa (*Low-rate wireless private area networks – LR-WPAN*) e fornece suporte a três bandas de canais de frequência utilizando o método DSSS (*Direct Sequence Sread Spectrum*) (AL-FUQAHA et al., 2015).

O padrão IEEE 802.15.4 suporta dois tipos de nós de rede: dispositivo de função completa (*Full Function Device - FFD*) e dispositivo de função reduzida (*Reduced Function Device - RFD*). Os FFDs podem funcionar como nós normais ou coordenadores de

rede de área pessoal (*Personal Area Network - PAN*) com responsabilidades para criar, controlar e manter a rede. Por outro lado, os RFDs são apenas nós simples, com recursos limitados e que só podem se comunicar com coordenadores (AL-FUQAHA et al., 2015).

Uma rede IEEE 802.15.4 pode assumir três topologias: estrela, ponto-a-ponto e árvore de *clusters* (*clusters-tree*). A topologia em estrela contém pelo menos um FFD no centro e alguns nós RFD. A topologia ponto-a-ponto contém um coordenador e os outros nós comunicam-se entre eles através de nós intermediários para outras redes. Por sua vez, a topologia em árvore de *clusters* constitui um caso especial da topologia ponto-a-ponto constituido por coordenador, *cluster head* e nós normais (AL-FUQAHA et al., 2015).

Electronic Product Code (EPCglobal)

O Electronic Product Code (EPCglobal) (AL-FUQAHA et al., 2015) possui uma organização responsável pelo seu desenvolvimento, com um número único de identificação armazenado em uma etiqueta RFID (*Radio Frequency Identification*). A arquitetura de rede EPCglobal é considerada promissora para IoT devido à escalabilidade, interoperabilidade, confiabilidade e suporte aos requisitos primários da IoT (AL-FUQAHA et al., 2015). Esta arquitetura usa o modelo de solicitação/resposta para a troca de dados entre dispositivos (TOLCHA; KIM, 2016).

Os EPCs são classificados em quatro tipos: 96-bit, 64-bit (I), 64-bit (II) e 64-bit (III). EPCs de 64-bit têm suporte para 16 mil companhias com identidades únicas e cobrem até 9 milhões de produtos e 33 milhões de números de série para cada. Por outro lado, os de 96-bit têm suporte para cerca de 268 milhões de companhias, 16 milhões de classes de produtos com 68 bilhões de números de série para cada classe (AL-FUQAHA et al., 2015).

O sistema RFID é dividido em dois componentes principais: o *transponder* do sinal de rádio ou *tag* e o leitor de *tag*. A rede EPCglobal pode ser dividida em cinco componentes: EPC, sistema de identificação, *middleware* EPC, descoberta de serviços e serviços de informação do EPC (AL-FUQAHA et al., 2015).

O sistema de identificação vincula as identidades EPC a um banco de dados usando leitor EPC através do *middleware* e o serviço de descoberta encontra dados necessários pelas *tags* usando o (*Object-Naming Services*) (AL-FUQAHA et al., 2015).

Long Term Evolution-Advanced (LTE-A)

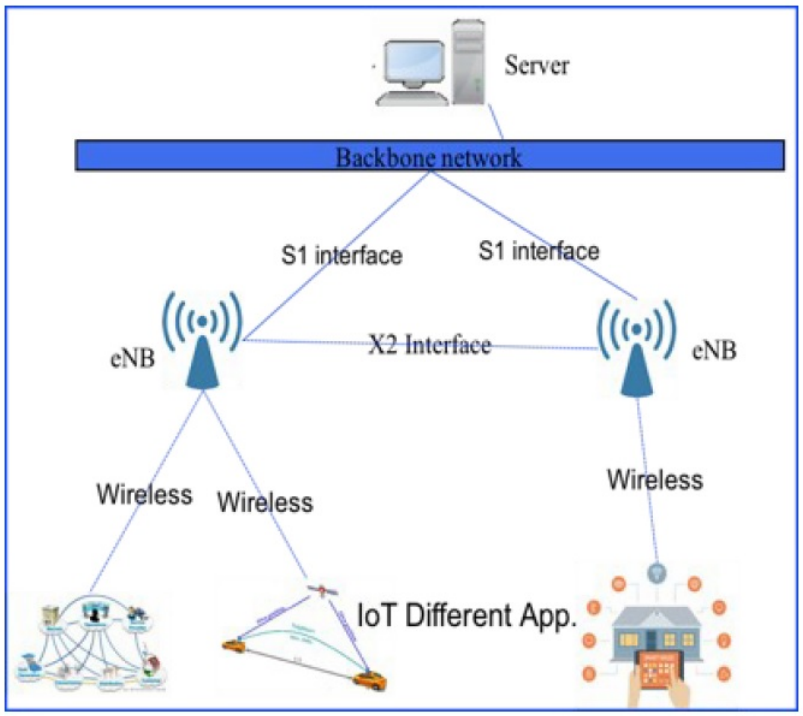
O Long Term Evolution-Advanced (LTE-A) (SALMAN; JAIN, 2019) é uma coleção de padrões de rede celular projetada para atender aos requisitos de comunicações M2M e IoT. Ela usa a tecnologia de Acesso Múltiplo por Divisão de Frequência Ortogonal (*Orthogonal Frequency Division Multiple Access - OFDMA*) para acesso ao meio (SALMAN; JAIN, 2019). A tecnologia OFDMA divide a largura de banda do canal em bandas menores chamadas blocos de recursos físicos (*Physical Resource Blocks - PRB*). Ela pode empregar a técnica de espalhamento de espectro de componentes múltiplos (*multiple-*

component-carrier (CC) spread spectrum) que permite ter até cinco bandas de 20 MHz (AL-FUQAHA et al., 2015).

A arquitetura do LTE-A depende de três partes essenciais: uma rede central (*Core Network - CN*), uma rede de acesso por rádio (*Radio Access Network – RAN*) e os nós móveis (SALMAN; JAIN, 2019; VAIGANDLA; KARNE; RAO, 2021). A rede central tem a função de controlar os dispositivos móveis e os seus endereços de IP. A rede de acesso por rádio lida com comunicações sem fio e acesso por rádio, estabelecendo protocolos de planos de usuário e de controle (AL-FUQAHA et al., 2015; SALMAN; JAIN, 2019).

A rede de acesso por rádio consiste em estações base, também designadas por envolved NodeBs (eNB), nas quais nós móveis são conectados usando comunicações sem fio. Assim, as estações base são conectadas entre si usando interface X2 e a rede central é conectada à rede de acesso por rádio através da interface S1 (AL-FUQAHA et al., 2015; SALMAN; JAIN, 2019). A Figura 20 mostra a arquitetura LTE-A.

Figura 20 - Arquitetura do LTE-A



Fonte: (SALMAN; JAIN, 2019)

Z-Wave

O Z-Wave (LILLI et al., 2021) é um protocolo de comunicação sem fio de baixa potência baseado em rádio frequência, com frequências entre 865 e 926 MHz. Ele é usado principalmente em sistemas de automação residencial. Em uma rede Z-Wave, os dispositivos são gerenciados e controlados por um controlador que também atua como um *gateway* permitindo ao usuário interagir com dispositivos Z-Wave via Internet ou rede local (KIM et al., 2018).

O desenvolvimento do Z-Wave é controlado pela Z-Wave Alliance, que envolve mais de 600 empresas. Ele usa uma rede em malha que pode conectar até 232 nós (MARKSTEINER et al., 2017). Os nós podem ser controladores, que são responsáveis por incluir e excluir nós e manter a tabela de roteamento, ou trabalhadores que são as "coisas" inteligentes da rede como luzes, interruptores, sensores entre outros (LILLI et al., 2021).

Na sua arquitetura, o Z-Wave possui em quatro camadas: a física, a de transporte, a de rede e a de aplicação. A camada física gerencia a modulação e as atribuições de canal de rádio frequência (RF). A camada de transporte controla a transferência de dados entre dois nós, e é resposável pelo reconhecimento do quadro, validação de dados e retransmisão de dados. A camada de rede gerencia o roteamento de quadros, faz a descoberta de topologia e mantém a tabela de roteamento. A camada de aplicação é responsável pelo controle de *payloads* em quadros recebidos ou transmitidos (KIM et al., 2020).

A segurança do Z-Wave é alcançada criptografando mensagens trocadas entre os dispositivos. Na primeira versão da camada de segurança (*S0 Security*) o Z-Wave usava uma chave temporária. A nova versão (*S2 Security*) fornece integridade de mensagens, confidencialidade, autenticação e atualização de dados, combinando criptografia simétrica e código de autenticação de mensagem (LILLI et al., 2021).

A Tabela 2 resume as principais características dos protocolos de infraestrutura.

Tabela 2 - Principais características dos protocolos de infraestrura.

Protocolos de infraestrutura	Características	Técnica de Propagação	Banda de Rádio (MHz)	Acesso ao meio	Taxa de dados	Estabilidade
RPL	Organiza a rede física em forma de um grafo acíclico direcionado	-	-	-	-	-
6LoWPAN	Suporta *unicast, multicast e broadcast*. Permite comunicação escalável e com mobilidade entre dispositivos	-	-	-	-	-
IEEE 802.15.4	Baixo consumo de energia, baixa taxa de dados, confiabilidade	DSSS	868/915/2400	TDMA, CSMA/CA	20/40/250 K	65 K nós
BLE	Foco em comunicações sem fio de baixa potência e de curto alcance. Baixo consumo de energia	FHSS	2400	TDMA	1024 K	5917 slaves
EPCglobal	Escalável, interoperável, confiável e suporta os requisitos primários da IoT	DS-CDMA	860 960	ALOHA	Varia de 5 640 K	-
LTE-A	Coleção de padrões para M2M. Divide a banda em blocos PRB	Multiple CC	Varia	OFDMA	1G (up), 500M (down)	-
Z-Wave	Usado em omunicações sem fio de baixa potência. Baseado em RF e usado principalmente para automação residencial	-	868/908/2400	CSMA/CA	40 K	232 nós

Fonte: Adaptado de (AL-FUQAHA et al., 2015)

2 SIMULADORES, *TESTBEDS* E EMULADORES

Quando um novo sistema é desenvolvido, torna-se imprescindível avaliar o seu comportamento e o desempenho (ALMUHAYA et al., 2022). No contexto de IoT esta prática também é recomendada, uma vez que a implantação pode gerar vários problemas inesperados, como falha de um nó ou desconexão da rede. Devido aos desafios de IoT, relacionados às limitações de hardware dos nós dos sensores ou ambiente no qual os nós são implantados é necessário verificar os protocolos em cada estágio de projeto, desenvolvimento e implementação (PAPADOPOULOS et al., 2017).

A verificação pode ser feita aplicando a modelagem e simulação computacional. Um modelo de simulação consiste em um projeto que resume o comportamento e o desempenho dos estudos de caso de um sistema, considerando os algoritmos de computação, termos físicos e matemáticos (ALMUHAYA et al., 2022). Portanto, simuladores podem ser utilizados antes da implementação do sistema no mundo real (PAPADOPOULOS et al., 2017).

Os simuladores fornecem prova de conceito para novas soluções em um ambiente virtual, evitando, desta forma, as experimentações do mundo real, que podem ser demoradas e custosas. No entanto, usando simuladores e modelos formais é difícil reproduzir com precisão as especificidades do ambiente de implantação. Assim, a adição de experimentação utilizando *testbeds* pode melhorar a avaliação de desempenho. Os *testbeds* são plataformas para experimentação de projetos de desenvolvimento que permitem experimentos replicáveis de novos modelos e tecnologias (PAPADOPOULOS et al., 2013; PAPADOPOULOS et al., 2017).

Através de *testbeds*, pesquisadores têm um amplo escopo de ambientes IoT para criar, desenvolver, depurar, solucionar problemas e analisar os seus modelos. No contexto de IoT, os *testbeds* são usados para criar aplicativos sem usar dispositivos físicos. Neste caso, o sistema de controle é conectado ao seu teste e o sistema potencial é programado para monitorar todos os dispositivos (PATEL; MEHTRE; WANKAR, 2019).

Para preencher a lacuna existente entre a simulação e experimentação, de modo que o sistema ou aplicação permaneça o mais próximo possível das condições de programação de sistemas embarcados reais, foram projetados os emuladores (PAPADOPOULOS et al., 2017).

Segundo (PATEL; MEHTRE; WANKAR, 2019), "um emulador é um programa de software ou hardware que permite que um sistema se comporte como outros dispositivos de computação para teste". Os emuladores estão num meio-termo entre a simulação real e os *testbeds* sem fio.

Os emuladores recorrem ao mesmo código para simulações e experimentação, permitindo que os usuários conheçam as especificidades da aplicação ou sistema desde a

configuração da simulação. No entanto, pode não haver precisão na modelagem de alguns parâmetros relativos ao ambiente de rádio e comportamento do nó mesmo que o código permaneça o mesmo (PAPADOPOULOS et al., 2017).

A seguir, estão descritos alguns simuladores, *testbeds* e emuladores utilizados no contexto de IoT.

2.1 Simuladores

Esta seção descreve brevemente treze simuladores selecionados no contexto de IoT: DPWSim, iFogSim, CupCarbon, SimIoT, OMNeT++, NS-Series, QualNet, Atomition IoT Simulator, SWE Simulator, MobIoTSim, TOSSIM e Bevywise.

DPWSim

O Device Profile for Web Services Simulator (DPWSim) é um kit de ferramentas que permite simular dispositivos e protocolos DPWS. Com o DPWSim desenvolvedores podem criar protótipos, desenvolver e testar aplicações e serviços de IoT sem a necessidade de ter dispositivos físicos. Ele simula um ambiente criando dispositivos virtuais que podem ser descobertos na rede e podem se comunicar com outros dispositivos ou clientes via protocolos DPWS. Ele é mais indicado quando o objetivo é se concentrar na lógica do negócio das aplicações em vez do desempenho físico dos dispositivos (TSOUTSA; FITSILIS; RAGOS, 2018). As principais habilidades do DPWSim envolvem: (i) independência de plataforma devido a JVM (*Java Virtual Machine*); e (ii) gerenciamento de dispositivos virtuais e suporte para os protocolos SOAP e HTTP (MARKUS; KERTESZ, 2020).

iFogSim

O iFogSim é um simulador de *edge* e *fog computing*, capaz de simular *fog data centers*, *edge devices*, conexões de rede e análise na nuvem. Ele é capaz de avaliar as políticas de gerenciamento de recursos aplicáveis aos ambientes de névoa (*fog*) com relação à latência, consumo de energia, congestionamento de rede, custos operacionais, largura de banda, agendamento de tarefas entre outros (GUPTA et al., 2017; PATEL; MEHTRE; WANKAR, 2019). Suas métricas de desempenho são fornecidas através do modelo *Sense-Process-Actuate*. Os sensores publicam dados obtidos periodicamente ou baseada em eventos. As aplicações dos *fog devices* se inscrevem e processam os dados provenientes dos sensores. Os resultados são executados pelos atuadores (GUPTA et al., 2017).

CupCarbon

O CupCarbon é um simulador para redes de sensores sem fio cujos objetivos são o projeto, visualização 2D/3D e validação de algoritmos para monitoramento ambiental, coleta de dados entre outros. Ele permite criar cenários com estímulos ambientais como incêndios, gases ou com objetos móveis em projetos educacionais ou científicos. O Cup-

Carbon oferece dois ambientes de simulação, um que possibilita cenários com mobilidade e geração de eventos naturais e o outro que possibilita a simulação de eventos discretos em redes de sensores sem fio (LOPEZ-PAVON; SENDRA; VALENZUELA-VALDÉS, 2018).

Este simulador permite que a rede projetada use o OpenStreetMap (OSM), inserindo diretamente os sensores com configuração individualizada na linguagem SenScript. O CupCarbon não implementa todas as camadas de protocolo. Por isso, a sua função é complementar os simuladores existentes. Pode simular os protocolos ZigBee, LoRa e WiFi (LOPEZ-PAVON; SENDRA; VALENZUELA-VALDÉS, 2018).

SimIoT

O SimIoT é um *kit* de ferramentas usado em ambiente IoT para simular o recurso e falha de função de aquiescência estática/dinâmica ou multiusuário em tempo real. O SimIoT é uma extensão do SimIC com uma camada extra para comunicação entre entidades. A sua arquitetura possui três camadas principais: nível de usuário, nível SimIoT e nível SimIC. As tarefas são executadas em máquinas virtuais, com retrição de largura de banda para controlar a carga da rede (MARKUS; KERTESZ, 2020), (RASHID; SHARIF; SHREEF, 2022).

OMNeT++

O OMNeT++ é uma ferramenta de código aberto usado para simulação de eventos discretos. Ele fornece mecanismo para modelar e simular qualquer tipo de rede, com blocos de construção de nós. Os blocos de nós modelam seus comportamentos e interações, que são usados no campo da simulação de rede (XIE; LI; GAO, 2020).

NS-Series

Network Simulator (NS-Series) é uma sequência de vários simuladores de rede: NS-1, NS-2, NS-3 e NS-4. Eles suportam vários protocolos como FTP, TCP, UDP e DSR. Os NS-Series possuem software livre, autorizados sob a licença GNU GPLv2 (PATEL; MEHTRE; WANKAR, 2019; XIE; LI; GAO, 2020).

QualNet

O QualNet é uma versão comercial do GloMoSim (*Global Mobile Information System Simulator*), que fornece diretrizes para redes do padrão IEEE 802.15.4. O QualNet dá suporte à simulação de vários dispositivos de rede com maior precisão, oferece portabilidade de plataforma, podendo ser executado em vários sistemas operacionais (AIREEN et al., 2017; PATEL; MEHTRE; WANKAR, 2019).

Atomiton IoT Simulator

O Atomiton IoT Simulator é um *framework* construído sobre o Atomition Stack que é usado para prototipagem e experimentos. Ele fornece suporte à simulação de sensores, atuadores e dispositivos virtuais com comportamentos exclusivos. Ele também permite testar escalabilidade em soluções de IoT, recorrendo a simulação de milhares de

dispositivos e eventos (DIAS et al., 2018; PATEL; MEHTRE; WANKAR, 2019).

SWE

O SWE Simulator é usado para projetar e verificar grandes redes de sensores. As ferramentas do SWE Simulator permitem verificar riscos de restrição e minimizar o custo de implementação. Com esse simulador é possível projetar uma ampla gama de redes de sensores (PATEL; MEHTRE; WANKAR, 2019).

TOSSIM

O TinyOS SIMulator (TOSSIM) é um simulador de eventos discretos que simula aplicações e dispositivos TinyOS em redes de sensores sem fio (DIAS et al., 2018; PATEL; MEHTRE; WANKAR, 2019). TinyOS é um sistema operacional baseado em componentes embutidos para dispositivos de baixa potência. O simulador TOSSIM faz parte do pacote TinyOS. Pelo fato do código no simulador ser usado para operar dispositivos reais, o TOSSIM pode ser considerado um emulador (AL-ROUBAIEY; AL-JAMIMI, 2019).

MobIoTSim

O MobIoTSim é um simulador implementado como uma aplicação móvel para o sistema operacional Android. Ele usa o protocolo MQTT e a biblioteca de código aberto Eclipse Paho para envio de dados. O MobIoTSim foi projetado para auxiliar pesquisadores a aprender o manuseio de dispositivos IoT sem a necessidade de ter sensores físicos. Ele possibilita realizar experimentos e demostrações de aplicações IoT com vários dispositivos. O sistema pode ser conectado a um *gateway* em nuvem, e, assim, gerenciar os dispositivos simulados e enviar de volta notificações respondendo a valores críticos do sensor (PFLANZNER et al., 2016; DIAS et al., 2018).

IoTIFY

O IoTIFY é um ambiente de desenvolvimento de aplicativos para IoT sem dependência de hardware, pois faz a virtualização de dispositivos. Ele fornece meios para a construção de modelagem analítica, prototipagem, experimentos e simulação de rede para dimensionamento do sistema e geração de dados (DIAS et al., 2018; PATEL; MEHTRE; WANKAR, 2019).

Bevywise

O Bevywise é um simulador para IoT que usa o protocolo MQTT como protocolo de roteamento. Ele permite simular vários dispositivos IoT e auxilia na análise do nível de desempenho da rede das aplicações, criando redes e dispositivos virtuais. O Bevywise pode ser usado para demonstração, desenvolvimento e teste em tempo real. As informações de simulação podem ser armazenadas em registros FLAT ou bancos de dados MySQL e SQLite (PATEL; MEHTRE; WANKAR, 2019; MANIVANNAN; RADHAKRISHNAN, 2020; SINGH; NANDANWAR; CHAUHAN, 2022).

A Tabela 3 apresenta um resumo com as principais características dos simuladores.

Tabela 3 - Principais características dos simuladores.

Simulador	Escopo	Tipo	LP	CA IoT	PIoT	IAPI	Licença	Segurança
DPWSim	IoT	-	Java	Aplicação	Secure Web Seviv-ces Messaging	SOAP	Aberta	Média
iFogSim	Fog	Eventos discretos	Java	Percepção, Rede e Aplicação	Não	SOAP	Aberta	Média
CupCarbon	Rede	Eventos discretos	Java/Custom Scripting	Percepção e Rede	802.15.4 e Lo-RaWAN	UDX	Aberta	Alta
SimIoT	Análise de da-dos	Eventos discretos	Java	Aplicação	Não	REST	Aberta	Alta
OMNeT++	Rede	Eventos discretos	C++	Percepção e Rede	Extensão manual	SOAP	Académica	Média
N-Series	Rede	Eventos discretos	C++	Percepção e Rede	802.15.4 e Lo-RaWAN	REST	Aberta	Alta
QualNet	Rede	Eventos discretos	C++	Percepção e Rede	802.15.4 (Somente Zigbee)	REST	Fechada	Média
Atomiton IoT Simulator	IoT e IIoT	Edge	Go/Java	Comunicação e Rede	-	REST	Fechada	Alta
SWE Simulator	Rede de sen-sores	Serviço de observação de sensores	C/C++	Comunicação e Rede	detecção de coli-são	SOAP	-	Alta
TOSSIM	TinyOS	Serviço de observação de sensores	C++/Python	Comunicação e Rede	-	SOAP	Aberta	Alta
MobIoTSim	Redes IoT	Baseado em pes-quisa	C++/C#	Aplicação e Rede	Perfil de dispositi-vos para serviços Web (DPWS)	REST	Aberta	Média
IoTIFY	Conexão de hardware	Aplicação móvel	Python/Java	Aplicação e Rede	Tempo real	REST	Fechada	Alta
Bevywise	Dispositivos IoT	Broker	Python/Java	Rede	Tempo real	REST	Aberta	Média

Legenda: LP-Linguagem de Programação, CAIoT-Camadas da Arquitetura IoT, PIoT-Padrões de IoT integrados

Fonte: (CHERNYSHEV et al., 2017),(PATEL; MEHTRE; WANKAR, 2019), (SINGH; NANDANWAR; CHAUHAN, 2022), (RASHID; SHARIF; SHREEF, 2022), (MANIVANNAN; RADHAKRISHNAN, 2020)

2.2 *Testbeds*

Esta seção descreve cinco *testbeds* no contexto de IoT: MBTAAS, WHYNET, FIT IoT-LAB, FIESTA-IOT e JOSE.

MBTAAS

O Model-Based Testing as a Service (MBTAAS) é um modelo de teste que é oferecido como serviço. Ele permite experimentos sistemáticos em IoT e em plataformas de dados. Sua abordagem combina técnicas de teste baseado em modelo e soluções orientadas a serviços (DIAS et al., 2018).

WHYNET

O Wireless HYbrid NETwork (WHYNET) é uma plataforma de teste híbrida. Ela permite o uso de simulação, emulação e hardware real. O WHYNET é desenvolvido para protocolos de redes de sensores sem fio heterogêneos e permite a integração consistente de componentes físicos e de emulação em um único modelo (PATEL; MEHTRE; WANKAR, 2019), (SYED et al., 2020).

FIT IoT-LAB

O Federated IoT testbed for large scale experimentation (FIT IoT-LAB) é um *testbed* de larga escala e de acesso aberto operando em seis *sites* diferentes na França. A plataforma possui um grande número de nós distribuídos que fornecem aos usuários várias ferramentas científicas para projetar, desenvolver elementos relacionados a IoT. Os nós deste *testbed* se comunicam sobre o protocolo IEEE 802.15.4. Além dos nós, a plataforma disponibiliza robôs programáveis para o ambiente de teste, que fornecem mobilidade em um modelo semelhante a um circuito. Há possibilidade dos usuários integrarem *hardware* personalizado em seus experimentos (CHERNYSHEV et al., 2017; PATEL; MEHTRE; WANKAR, 2019; SINGH; NANDANWAR; CHAUHAN, 2022).

FIESTA-IOT

FIESTA-IoT é uma plataforma federada com 10 *testbeds* para aplicações IoT interoperáveis. Este conjunto de *testbeds* oferece serviços da Web em tempo real sobre conjuntos de dados IoT. Dessta forma, ela permite experimentação de aplicações IoT em escala global (PATEL; MEHTRE; WANKAR, 2019).

A plataforma fornece uma infraestrutura experimental de projeto, ferramentas de software, técnicas semânticas, processos de certificação e melhores práticas que permitem a interconexão de recursos de forma semântica interoperável (SERRANO et al., 2018). O seu objetivo principal é abrir novos horizontes para o desenvolvimento e implantação de aplicações e experimentos no contexto de IoT, com base na conexão e interoperabilidade de diversas plataformas e *testbeds* (SERRANO et al., 2018). Ele aborda questões de interoperabilidade semântica em sete níveis: (i) nível de hardware; (ii) nível de dados; (iii) nível de modelo; (iv) nível de consulta; (v) nível de raciocínio; (vi) nível de serviço ou

nível de aplicação; e (vii) nível de domínio de aplicação (GYRARD; SERRANO, 2015).

JOSE

O Japan-Wide Orchestrated Smart/Sensor Environment (JOSE) é um *testbed* que está localizado no Japão, oferecido como serviço de rede virtualizado para avaliação de serviços de IoT. A plataforma JOSE oferece a possibilidade de fazer experimentos em larga escala que consistem em experimentos de armazenamento, redes de sensores e outros recursos de computação (PATEL; MEHTRE; WANKAR, 2019).

Segundo (CHERNYSHEV et al., 2017) a plataforma suporta a execução simultânea de vários serviços de IoT, cada um com sua própria rede de sensores físicos e infraestrutura de nuvem virtual na camada de aplicação. A infraestrutura da camada de aplicação está espalhada por cinco centros de dados no Japão.

JOSE permite uma experimentação flexível na camada de aplicação da infraestrutura. Ele fornece recursos para gerenciamento e personalização da infraestrutura virtual específica do serviço e redes definidas por software (*software-defined networking* - SDN). Além disso, os sensores são compatíveis com IEEE 1888 que coletam dados ambientais usando protocolos IEEE 802.11, 3G ou LTE (CHERNYSHEV et al., 2017).

2.3 Emuladores

Esta seção descreve brevemente quatro emuladores selecionados no contexto de IoT: Cooja, MAMMotH, NetSim e NCTUns.

Cooja

O Cooja é um simulador baseado em rede, focado nas aplicações executadas no sistema operacional Contiki e rede de sensores sem fio. Cooja emula o firmware que é carregado para os nós de névoa (*fog*) e alternativamente simula dispositivos IoT (PATEL; MEHTRE; WANKAR, 2019; SINGH; NANDANWAR; CHAUHAN, 2022). Ele suporta a maioria dos padrões e protocolos implementados pelo sistema operacional Contiki, o que possibilita replicar cenários reais com a incorporação de protocolos de aplicação mais usados. Para o roteamento, o Cooja usa IPv4 ou IPv6 e suporta topologia de rede dinâmica (SINGH; NANDANWAR; CHAUHAN, 2022).

MAMMotH

O MAMMotH é um emulador IoT com capacidade de emular milhares de dispositivos por máquina virtual. A sua arquitetura pressupõe a existência de três cenários: (i) dispositivos móveis conectados via GPRS a uma estação base, formando uma topologia em estrela; (ii) uma rede de sensores sem fio conectada a uma estação base via GPRS; e (iii) dispositivos restritos que se conectam a *proxies* conectados ao *back-end* de um emulador IoT de grande escala (DIAS et al., 2018; PATEL; MEHTRE; WANKAR, 2019).

NetSim

O Network Simulator (NetSim) é um simulador e emulador de eventos discretos usado para modelagem ou simulação de protocolos, dispositivos de rede e aplicações de segurança. O NetSim oferece um ambiente para testar uma ampla variedade de redes como Wi-Fi, WSN, Celular-GSM, LTE, entre outras (PATEL; MEHTRE; WANKAR, 2019; BAKARE; ENOCH, 2019).

O emulador NetSim fornece aos usuários a possibilidade de transmitir tráfego ao vivo e aprender o efeito de dispositivos reais. Ele implementa tarefas que podem não ser realizadas em laboratório, misturando o mundo virtual e o real. Possui interfaces de linha de comando e gráfica, que permitem aos usuários analisar, projetar e testar uma variedade de redes (BAKARE; ENOCH, 2019).

NCTUns

O National Chiao Tung University Network Simulator (NCTUns) é um emulador de rede, de código aberto e baseado em eventos discretos. Ele que fornece um ambiente de rede para teste e desenvolvimento (PATEL; MEHTRE; WANKAR, 2019).

O NCTUns oferece a possibilidade de avaliar vários aspectos relacionados com o gerenciamento de recursos de rádio ou esquemas de QoS em aplicações do mundo real. Ele também possui suporte integrado para DiffServ, WLAN, VoIP, GPRS, RTP/RTCP, malha sem fio e protocolos IEEE 802.11 (e) (TYAGI; DEMBLA, 2015).

3 TRABALHOS RELACIONADOS

Este capítulo aborda os principais trabalhos que analisam o desempenho de diferentes protocolos de aplicação em redes IoT e *ad hoc*. Os estudos analisados focam em métricas como latência, perda de pacotes, largura de banda e eficiência energética, utilizando diversos protocolos de comunicação e diferentes ambientes de teste. A seguir, os principais trabalhos relacionados são apresentados e comparados com a proposta desta dissertação.

O trabalho de Silva *et al.* (SILVA et al., 2021) avalia o desempenho dos protocolos MQTT, CoAP e OPC UA em dois ambientes de teste, local e de larga escala. Os autores utilizam a plataforma FIT-IoT na avaliação. As métricas analisadas foram latência e perda de pacotes. Os autores concluem que o CoAP apresenta a menor latência.

O artigo de Bansal e Kulmar (BANSAL; KUMAR, 2019) oferece uma visão geral dos protocolos MQTT, CoAP e DDS no contexto de cidades inteligentes. Os autores utilizam o software Contiki para simular diferentes cenários, além de considerar como métricas a latência, largura de banda e perda de pacotes. O artigo destaca que o CoAP apresenta menor latência comparado aos outros protocolos.

O trabalho de Macedo (MACEDO, 2019) analisa seis protocolos incluindo HTTP, MQTT, CoAP, AMPQ, XMPP e WebSocket, em um ambiente Wi-Fi com dispositivos NodeMCU Esp8266 e Android. As métricas avaliadas foram RTT, tamanho da mensagem, número de dispositivos e largura de banda.

O estudo de Ramirez e Pedraza (RAMIREZ; PEDRAZA, 2017) faz uma análise comparativa entre os protocolos MQTT, AMQP, CoAP e XMPP. O ambiente de execução utiliza um servidor virtual na Google Cloud e dispositivos móveis conectados a um Raspberry Pi. As métricas analisadas foram latência, tempo de envio, uso de CPU e quantidade de pacotes enviados.

Dutra *et al.* (DUTRA; MORAES; AMORIM, 2012) propuseram o protocolo Radnet (RP), realizando uma comparação com os protocolos RP, AODV e G3AODV em simulações e experimentos em laboratório. As métricas utilizadas foram taxa de entrega, sobrecarga de mensagens e latência, com o Radnet demonstrando melhor desempenho em cenários de redes móveis.

A Tabela 4 resume os principais aspectos dos trabalhos citados anteriormente. Na tabela são identificados os protocolos avaliados, o ambiente de teste e as métricas consideradas, além de destacar as diferenças relacionadas a esta dissertação, que foca no protocolo Radnet, introduz novos cenários de testes e métricas específicas (*Round Trip Time* (RTT), tamanho de mensagem, largura de banda, número de dispositivos, latência, taxa de entrega, *jitter*).

Tabela 4 - Tabela comparativa entre trabalhos relacionados e a pesquisa desta dissertação.

Autor(es)/Ano	**Protocolos**	**Ambiente de Testes**	**Métricas**	**Diferenças da Pesquisa**
Silva *et al.*, 2021	MQTT, CoAP, OPC UA	Local e plataforma FIT-IoT (larga escala)	Latência, perda de pacotes	Uso do protocolo Radnet, diferentes cenários e métricas
Bansal e Kumar, 2019	MQTT, CoAP, DDS	Simulação com Contiki, cidades inteligentes	Latência, largura de banda, perda de pacotes	Uso do protocolo Radnet, cenários *ad hoc* e diferentes métricas
Macedo, 2019	HTTP, MQTT, CoAP, AMPQ, XMPP, WebSocket	Rede Wi-Fi, dispositivos NodeMCU Esp8266 e Android	RTT, tamanho da mensagem, número de dispositivos, largura de banda	Uso do protocolo Radnet, diferentes cenários e métricas
Ramirez e Pedraza, 2017	MQTT, AMQP, CoAP, XMPP	Servidor virtual (Google Cloud), dispositivos móveis e Raspberry Pi	Latência, tempo de envio, uso de CPU, pacotes enviados	Uso do protocolo Radnet, diferentes cenários e métricas
Dutra *et al.*, 2012	RP, AODV, G3AODV, Radnet	Simulação com NS-3 e experimentação em laboratório (20 nós)	Taxa de entrega, sobrecarga de mensagens, latência	Esta dissertação complementa com novos cenários e a utilização de *linux network namespaces*

4 AMBIENTE DE EXECUÇÃO

Este capítulo apresenta o ambiente de execução proposto para a análise de desempenho dos protocolos selecionados. Cada uma das próximas seções define as métricas, como obter os ambientes de execução isolados utilizando a funcionalidade *Network Namespace*, os cenários dos experimentos e o ambiente propriamente dito, a aplicação do domínio da agricultura, a implemetação dos protocolos de comunicação CoAP, MQTT e Radnet e a descrioção dos experimentos realizados.

4.1 Métricas

Esta seção apresenta as métricas utilizadas como base na análise de desempenho, que são: RTT, tamanho de mensagem, largura de banda, número de dispositivos, latência, *jitter* e taxa de entrega. Cada uma destas métricas está descrita a seguir.

RTT

O *Round Trip Time* (RTT) define o tempo total para envio de uma mensagem pela rede de um nó de origem até um nó de destino, acrescentado do tempo da transmissão da mensagem de retorno até o nó de origem.

Tamanho de mensagem

Considerou-se na avalição dos protocolos o tamanho das mensagens, em quantidade de bytes, enviadas entre um cliente e o servidor. Nos diferentes cenários dos experimentos são utilizadas mensagens de 500 e 5000 bytes. A razão para escolha destes valores é a necessidade de estudar o comportamento de cada protocolo em cenários com mensagens com diferentes ordens de grandeza. São consideradas no estudo como mensagens menores as que possuem um tamanho que não ultrapassa o MTU (*Maximunm Transmission Unit*) de 1500 bytes padrão do TCP e mensagens maiores aquelas que ultrapassam este valor. Os termos mensagem e pacote são empregados de forma intercambiável.

Largura de banda

A largura de banda determina a medida de capacidade de transmissão de um meio. Ela indica a quantidade máxima de dados que podem trafegar em uma conexão. A largura de banda é medida em bits por segundo (bps), mas também podem ser utilizados seus múltiplos, como Kbps, ou Mbps, e assim por diante. Considerou-se nos experimentos cenários de 30 e 50 kbps pelo fato do estudo estar relacionado com redes que possuem restrições de recursos (redes IoT). Redes IoT, em alguns casos, podem possuir uma infraestrutura limitada, e valores de 30 e 50 kbps podem representar cenários realistas, podendo simular situações onde a largura de banda disponível é limitada, como em áreas

remotas ou em sistemas de comunicação com baixa taxa de transmissão.

Número de dispositivos

Nos experimentos, variou-se a quantidade de dispositivos em cenários de redes com 3, 5 e 10 dispositivos para analisar o comportamento dos protocolos. O número de dispositivos foi combinado com o tamanho de mensagens e largura de banda. Para simplificar, os termos dispositivo e nó são empregados de forma intercambiável.

Latência

A latência é o tempo do atraso na transmissão. Ela mostra o tempo que os dados demoram para serem transferidos pela rede, entre o nó de origem e o nó de destino.

Taxa de entrega

A taxa de entrega de mensagens é a média das mensagens recebidas pelos nós de destino dividida pelo número de mensagens enviadas pelos nós de origem. As mensagens duplicadas são descartadas.

Jitter

Jitter é a medida da variação nos intervalos de tempo entre pacotes consecutivos transmitidos em uma rede. Em uma transmissão ideal, os pacotes devem chegar com intervalos constantes. No entanto, devido as flutuações na rede, os pacotes podem sofrer atrasos variáveis, gerando *jitter*.

4.2 Linux Namespace

O Linux Namespace é uma funcionalidade do *kernel* do sistema operacional Linux que permite a criação de ambientes isolados, conhecidos como *namespaces*. O *namespace* pode ser visto como um isolamento lógico dentro do *kernel* do Linux, que faz o controle da visibilidade.

Eles permitem separar recursos do sistema operacional. Cada *namespaces* possui sua própria visão dos recursos. Os recursos podem ser processos, memória do sistema operacional, instruções privilegiadas da CPU, sistemas de arquivos, redes entre outros. Desta forma, é possível que diferentes processos executem em um ambiente isolado, sem que haja interferência entre os processos.

Como todos os controles são definidos no nível do processo, um *namespace* controla quais recursos do *kernel* um processo pode ver. Portanto, para aplicações executadas no espaço do usuário, o *kernel* assume o controle e executa suas instruções em nome da aplicação (JAIN, 2020).

Existem vários tipos de *namespaces*, sendo cada um responsável pelo isolamento de um recurso. A seguir, definimos alguns dos seus principais tipos (Man7 Org, 2024).
Mount Namespace (mnt): isola o sistemas de arquivos, permitindo que cada *names-*

pace tenha a sua própria hierarquia de sistemas de arquivos montados. Processos em diferentes *namespaces* podem ter diferentes visões do sistema de arquivos e são independentes entre eles.

PID Namespace (pid): isola o espaço de identificação de processos, especificamente a tabela de processos. Processos em um *namespace* não podem interferir ou ver processos de outros *namespaces*. Desta forma, cada processo tem a sua própria visão dos processos em execução criando ambientes de execução isolados.

Network Namespace (net): isola a pilha de rede, contendo suas próprias interfaces de rede, endereços IP, tabelas de roteamento e outros recursos de rede. Desta forma, podem ser criados diferentes ambientes de rede isolados, onde cada *namespace* possui a sua própria pilha de rede virtual.

IPC Namespace (ipc): isola recursos de comunicação entre processos, como filas de mensagens, semáforos e segmentos de memória compartilhada. Um *namespace* pode ter seus próprios recursos de comunicação, independentes dos outros *namespace*. Assim, são criados ambientes de execução isolados.

UTS Namespace (uts): isola o nome do sistema (*hostname*) e o seu domínio (NIS domain name). Desta forma, podem ser criados ambientes de execução isolados, onde cada processo tem seu próprio nome de *host*. Um *namespace* com seu nome de *host* é independente dos outros *namespaces*.

User Namespace (user): isola a identidade do usuário e do grupo do sistema. Cada *namespace* tem seu próprio conjunto de IDs de usuário e grupo, o que significa que os processos em um *namespace* podem ter identidades de usuário e grupo diferentes dos processos em outros *namespace*. Dentro de um *namespace* de usuário, um processo pode ter privilégios de *root* (UID 0) enquanto, fora do *namespace*, o mesmo processo é executado com um UID não privilegiado. Assim, podem ser criados ambientes isolados com permissões de usuário independentes.

Cgroup Namespace (cgroup): isola a visão dos grupos de controle (cgroups) para processos em execução dentro de um *namespace*. Cgroups permitem organizar processos em grupos hierárquicos para gerenciar e impor limites nos recursos do sistema como CPU, memória e I/O. Embora os *namespaces* de cgroup não sejam um tipo de *namespace* separado como os outros descritos anteriormente (PID, montagem, rede), eles estão relacionados ao conceito de isolamento de *namespace*. *Namespaces* de cgroup virtualizam a visão da hierarquia de cgroup, de modo que os processos em execução dentro de um *namespace* de cgroup têm uma visão diferente da hierarquia em comparação com os processos em execução no *host* ou em outros *namespaces*.

Time Namespace (time): isola o relógio do sistema e os recursos relacionados ao tempo. Ele fornece visualizações de tempo independentes para cada processo. Assim, o isolamento permite que os processos tenham tempos de sistema diferentes. Este fato, possibilita a manipulação do tempo para fins de teste ou simulação. Ele aproveita instâncias de relógio

do sistema separadas para cada *namespace*, o que garante que as alterações na hora do sistema em um *namespace* não afetem outros *namespaces* ou o sistema *host*.

É importante ressaltar que diferentes *namespaces* podem ser combinados para criar ambientes altamente isolados, como em *containers*, onde cada *container* pode ter seu próprio conjunto de recursos isolados. Este trabalho está baseado em *Network Namespaces* descrito a seguir.

4.3 Network Namespace

Um *namespace* de rede é logicamente uma cópia da pilha de rede, com seus próprios dispositivos de rede, tabelas de roteamento, tabelas *netfilter*, soquetes de rede e outros recursos de rede. Um *namespace* fornece a um *container* um conjunto separado de subsistemas de rede. Portanto, um processo dentro do *namespace* da rede vê diferentes interfaces de rede, rotas e *iptables*, separando a rede de *containers* da rede *host* (ROSEN, 2014; JAIN, 2020).

As seguintes regras simples são aplicadas ao *namespace* de rede: (i) um dispositivo de rede deve estar sempre associado à exatamente um *namespace* de rede; (ii) um soquete deve estar sempre associado à exatamente um *namespace* de rede; e (iii) cada *namespace* de rede requer pelo menos um dispositivo de rede de *loopback*. Este dispositivo é usado para comunicação *host-local*. As interfaces de *loopback* em cada *namespace* são separadas umas das outras. As portas locais abertas em um *namespace* não são visíveis na interface de *loopback* de outro *namespace* (SEMJONOV, 2020).

Em uma rede de *namespaces*, que é frequentemente utilizada em contextos de *container* e virtualização, os elementos ativos e passivos podem ser classificados com base em suas funções e características. A terminologia ativo e passivo não é tão comum neste contexto específico, como em redes de computadores tradicionais. Esta classificação é definida a seguir:

1. Ativos: os elementos ativos são aqueles que executam operações, gerenciam recursos e processam dados. Eles necessitam de execução contínua e gerenciamento. Alguns exemplos de elementos ativos são *containers*, orquestradores, *plugins* de rede, controladores de cgroups e serviços de rede;

2. Passivos: os elementos são considerados passivos se fornecem infraestrutura e recursos para os elementos ativos operarem. Eles não executam operações por conta própria, mas são essenciais para a estrutura de isolamento e gerenciamento. Alguns exemplos de elementos passivos são *namespaces*, volumes, interfaces de rede virtuais e objetos ConfigMaps.

Assim como nas redes de computadores tradicionais, ambos os tipos de elementos

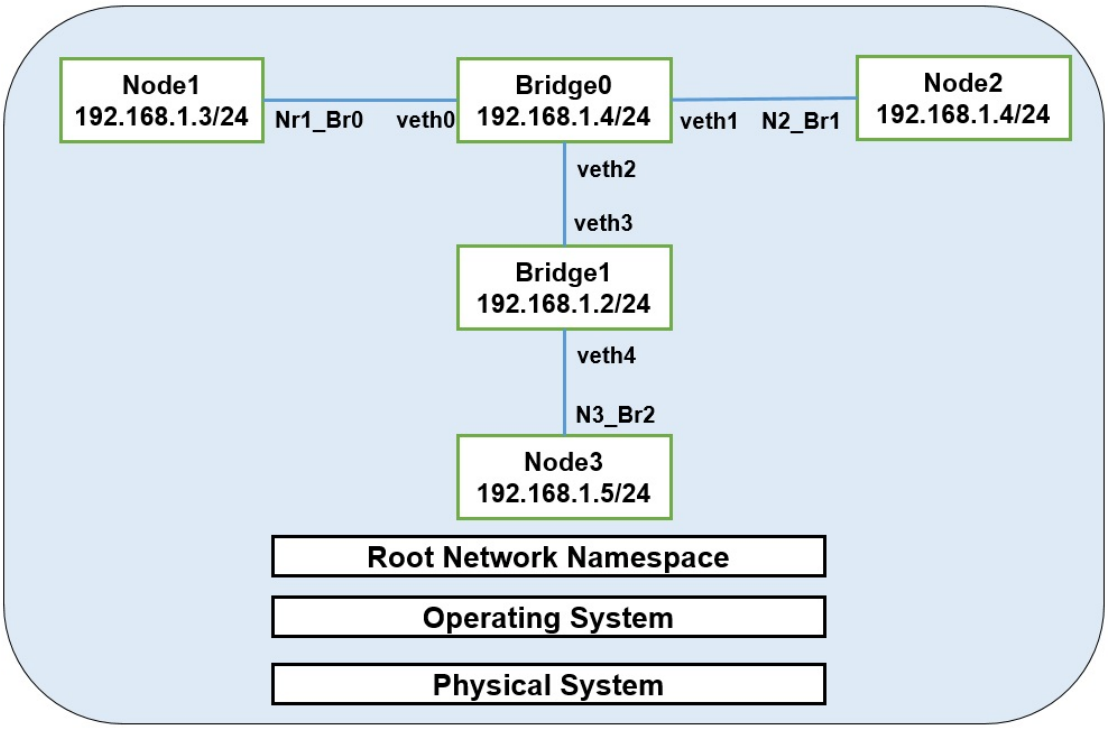
são essenciais em uma rede de *namespaces*. Os elementos ativos gerenciam, operam e garantem a funcionalidade, enquanto os elementos passivos fornecem a estrutura e os recursos necessários para a operação eficiente e segura dos elementos ativos.

4.4 Cenários dos Experimentos

Inicialmente foi projetada a avalição de desempenho dos protocolos usando um simulador, o que facilitaria a simulação de um número elevado de dispositivos, sensores e atuadores, com custos e riscos reduzidos. Entretanto, o grande problema constatado é a indisponibilidade de simuladores *open source* que suportam todos protocolos que seriam avaliados. Apesar da maior parte de simuladores oferecer APIs para criação de blibliotecas ou *plugins* para dar suporte a outros protocolos, devido a complexidade da arquitetura e tempo de desenvovimento, esta opção se mostrou inviável. Portanto, optou-se por realizar os experimentos usando os *namespaces* de rede.

Os experimentos foram executados em três cenários de redes baseados em *namespaces* com 3, 5 e 10 nós, cada. A Figura 21 ilustra o cenário de uma rede com três *namespaces* e duas *bridges*. Os cenários restantes, com 5 e 10 nós, seguem a mesma topologia.

Figura 21 - Rede de Namespaces criada.



Fonte: Autor

Como pode ser observado na Figura 21 existem três *namespaces*, `Node1, Node2` e

`Node3`, e duas *bridges*, `Bridge0` e `Bridge1`. O *namespace* `Node1` está conectado à *bridge* `Bridge0` pelas interfaces virtuais `N1_Br0` e `Veth0`. O `Node2` está conectado à mesma *bridge* pelas interfaces `N2_Br0` e `Veth1`. O `Node3` está conectado conectado à *bridge* `Bridge1` pelas interfaces `N3_Br2` e `Veth4`. As *bridges* estão conectadas entre si pelas interfaces `Veth2` e `Veth3`. A figura também apresenta a abstração do *namespace* raiz, o sistema operacional e máquina na qual a rede de *namespaces* foi criada.

Nos experimentos, para cada cenário de rede são combinados a quantidade de dispositivos (3, 5 ou 10), a largura de banda (30 kbps ou 50 kbps) e o tamanho das mensagens (500 bytes ou 5000 bytes). Para realizar a análise de desempenho são coletadas como métricas o RTT, a latência e a taxa de entrega de pacotes para cada um dos protocolos de comunicação CoAP, MQTT e Radnet.

Para a realização dos experimentos com os protocolos CoAP e MQTT e Radnet, foi necessário desenvolver módulos de software específicos para cada um deles. Foi utilizada a linguagem de programação Python, que permite a construção fácil destes módulos que podem ser executados através de *namespaces* das redes criadas. Porém, para o protocolo RadNet as aplicações foram implementadas na linguagem C, pois até ao momento é a única linguagem suportada pelo protocolo.

4.5 Ambiente dos experimentos

O ambiente dos experimentos é composto por elementos de hardware e software (bibliotecas, *broker* e ferramentas) que estão apresentados a seguir.

1. Hardware:
   (a) Computador/Servidor: Processador Intel Core i7-4790 CPU @ 3.60GHz de 8 núcleos, 8G de Memória RAM 1600 MHz SDRAM DDR3, Disco rígido 1TB mSATA SSD, Interface de rede Ethernet 10/100/1000 Mbps com Sistema Operacional Ubuntu 22.04.4 LTS de 64-bit.
2. Software:
   (a) Bibliotecas
      i. **paho-mqtt** (Eclipse Foundation, 2021): é uma biblioteca desenvolvida pela Eclipse Foundation que fornece implementações de clientes MQTT em várias linguagens de programação, incluindo C, Java, Python, JavaScript e outras. Algumas características desta biblioteca são a leveza, suporte multiplataforma, flexibilidade e confiabilidade.
      ii. **aiocoap** (AMSüSS; CONTRIBUTORS, 2013–): é uma biblioteca que implementa o Constrained Application Protocol (CoAP). O CoAP é um pro-

tocolo de comunicação especializado em dispositivos com recursos limitados, usado frequentemente em aplicações de Internet das Coisas (IoT). Ele é escrito em Python 3, e usa seus métodos asyncio nativos para facilitar operações simultâneas, mantendo uma interface fácil de usar. A biblioteca aiocoap permite a implementação de clientes e servidores.

iii. **scapy** (Scapy Community, 2024): é uma ferramenta de manipulação de pacotes para redes de computadores escrita em Python. Originalmente desenvolvida por Philippe Biondi, esta ferramenta é um programa interativo de manipulação de pacotes e uma biblioteca que pode ser usada para interagir com os pacotes na rede. Possui várias funcionalidades como a extração de dados acima da camada TCP usando a sua camada Raw e análise de protocolos específicos usando extensões como scapy-http. Portanto, o Scapy pode ser considerado tanto uma ferramenta quanto uma biblioteca.

iv. **netifaces** (Pypi Org, 2021): é uma biblioteca portátil em Python mantida pela Pypi Org, que enumera interfaces de rede na máquina local. Historicamente, tem sido difícil obter de forma direta os endereços de rede da máquina na qual *scripts* Python estão sendo executados sem comprometer a portabilidade do seu *script*. Portanto, com a netifaces torna-se possível fazer a enumeração de interfaces e endereços de rede preservando a portabilidade.

(b) *Broker*

i. **mosquitto** (Mosquitto Org,): é um *broker* MQTT de código aberto, projetado para facilitar a comunicação entre dispositivos em uma rede, especialmente em aplicações de Internet das Coisas (IoT). Mosquitto é um projeto *open source* mantido pela Eclipse Foundation.

(c) Ferramentas

i. **wireshark** (Wireshark Org,): é uma ferramenta de código aberto amplamente utilizada para análise de protocolos de rede. Permite capturar e inspecionar os dados trafegando em uma rede em tempo real, sendo essencial para diagnóstico de problemas, análise de desempenho e segurança em redes de computadores.

ii. **tcpdump** (Tcpdump Org,): é uma ferramenta de linha de comando usada para capturar e analisar pacotes de rede. É amplamente utilizada por administradores de rede e segurança para diagnosticar problemas de rede, monitorar tráfego e analisar dados de comunicação em redes IP. Desenvolvido originalmente por Lawrence Berkeley National Laboratory, tcpdump

é uma ferramenta poderosa que permite a inspeção detalhada do tráfego de rede.

iii. **iperf** (DUGAN et al., 2003-): é uma ferramenta de teste de rede que pode criar fluxos de dados TCP e UDP e medir a vazão, a largura de banda e a qualidade de uma conexão de rede. Desenvolvida para ser uma ferramenta poderosa e flexível, o Iperf permite experimentos de desempenho de rede em várias plataformas. Ele é executado com CLI (*Command Line Interface*) e permite a definição de vários parâmetros que podem ser usados para testar ou como alternativa, otimizar ou ajustar uma rede. Ele possui uma funcionalidade de cliente e servidor e pode medir a vazão entre as duas extremidades, de forma unidirecional ou bidirecional. O Iperf é *open source*.

iv. **iproute2** (Linux Foundation, 2023): é uma ferramenta desenvolvida para administradores de sistemas e redes. Ela fornece controle sobre todos os aspectos das configurações de rede em sistemas Linux. O iproute2 possui um conjunto de utilitários de rede para configuração, gerenciamento de interfaces de rede, tabelas de roteamento, controladores de tráfego e *namespaces* de rede.

4.6 Aplicação de IoT no domínio da agricultura

É inquestionável que a agricultura é uma atividade que desempenha uma função preponderante no desenvolvimento econômico de vários países. A demanda por qualidade dos alimentos e a necessidade de gerenciamento de impacto ambiental na agricultura levam à utilização da tecnologia de informação e comunicação no setor agrícola.

A IoT traz uma grande revolução no setor agrícola. Espera-se que ao aplicar a IoT, agricultores e pesquisadores consigam solucionar problemas como escassez de água, gestão de custos e problemas de produtividade. Portando, é necessário fornecer soluções para estes problemas buscando aumentar a produtividade e reduzir os custos (FAROOQ et al., 2019).

A IoT possibilita o monitoramento de vários parâmetros de produção, visando o alcance de melhores resultados. Portanto, é possível fazer o monitoramento da umidade do solo, de condições climáticas, da fertilidade entre outros parâmetros. As soluções tecnológicas para IoT podem ser classificadas em quatro domínios principais: monitoramento, controle, logística e previsão (FAROOQ; AKRAM, 2021). Cada um destes domínios estão descritos a seguir.

1. **Monitoramento:** trata da identificação de fatores ambientais e físicos coletados

em condições naturais, como culturas e campos usando as redes de sensores sem fio (RSSF). Os dados são recuperados sem a intervenção humana e transmitidos a um centro de dados para controle ou visualização. Neste domínio destacam-se os seguintes subdomínios: monitoramento do solo; monitoramento de temperatura; monitoramento da água; e monitoramento de plantas.

2. **Controle:** este domínio utiliza dispositivos atuadores como bombas, válvulas, umidificadores e alarmes. O principal objetivo deste domínio é melhorar o uso de água, pesticidas ou fertilizantes, coleta de informações de previsão do tempo e das RSSF. O controle auxilia na redução de custos. Neste domínio destacam-se os seguintes subdomínios: controle de fertilizantes e pesticidas; controle de irrigação; e controle de acesso de intrusos.

3. **Previsão:** o domínio de previsão fornece ferramentas e informações aos agricultores para a tomada de decisão. Por exemplo, uma ferramenta que recebe dados de umidade do solo capturados pelos sensores, analisa as informações e envia aos agricultores para medições adicionais ou tomada de decisão. A umidade do solo pode ser controlada através do resultado medido e o agricultor pode prever a produção da colheita. Neste domínio destacam-se os seguintes subdomínios: estimativa de produção, previsão de crescimento da colheita e previsão de condições ambientais.

4. **Logística:** este domínio trata de toda cadeia de valor. Ele engloba a produção agrícola, compras, expedição, armazenamento, inventário e colocação, embalagem, distribuição e atividades relacionadas. Portanto, a logística não só fornece informações transparentes e eficientes ao setor empresarial, mas também reduz o custo dos produtos agrícolas, protege as necessidades dos agricultores e aumenta a produtividade agrícola. Neste domínio destacam-se os seguintes subdomínios: produção, comercialização e transporte.

Nos experimentos, foram desenvolvidas aplicações da área de agricultura que simulam sensores de temperatura e umidade. Para o caso do MQTT foram desenvolvidas aplicações do lado cliente (subscritores e publicadores) que se comunicam com o *broker* `mosquitto`. Por sua vez, para o CoAP, foram desenvolvidas aplicações do lado cliente e do lado servidor, onde os clientes fazem solicitações ao servidor pelo método GET, que responde aos clientes enviando as informações sobre a temperatura e umidade geradas pelos sensores. Para o protocolo RadNet, também, foram desenvolvidas aplicações que permitem a comunicação dos sensores baseando-se no interesse destas aplicações.

4.7 Protocolos de comunicação

CoAP

O Código 4.1 apresenta a aplicação do cliente CoAP desenvolvida com a biblioteca *aiocoap*. A linha 7 define o método `GET` para a requisição feita ao recurso `agr`, na máquina local. Outras opções de métodos são `POST, PUT, DELETE`. Nas linhas numeradas de 11 a 13 está definida uma excepção para verificar se o cliente conectou ou não com o servidor. Caso ocorra a conexão é mostrada a resposta enviada pelo servidor.

```
1  import asyncio
2  from aiocoap import *
3
4  async def fetch():
5      protocol = await Context.create_client_context()
6
7      request = Message(code=GET, uri='coap://192.168.1.3/agri')
8      try:
9          response = await protocol.request(request).response
10         print('Resposta do Servidor: %s\n%r' % (response.code, response.
       payload))
11     except Exception as e:
12         print('Falha ao obter resposta do servidor CoAP: %s' % e)
13     finally:
14         await protocol.shutdown()
15
16 if __name__ == "__main__":
17     asyncio.run(fetch())
```

Código 4.1 - Exemplo de cliente CoAP.

Assim como para o cliente, o desenvolvimento da aplicação do servidor utiliza a biblioteca *aiocoap*. O Código 4.2 mostra a aplicação do servidor, com a importação dos módulos usados, *random, asyncio* e *struct*. Nas linhas numeradas de 8 a 11 está a classe `AgriResource()`, responsável por gerar dados de sensores de temperatura e umidade. Estes dados são gerados pela função `generate-sensor-data()`, que está nas linhas numeradas de 12 a 17. Neste exemplo, a mensagem enviada tem tamanho igual a 500 bytes de dados. Como os dados de temperatura e umidade gerados não atigem 500 bytes, eles são empacotados na linha 15 e na linha 16 a mensagem é preenchida com dados aleatórios para atingir os 500 bytes desejados.

A configuração e a inicialização do servidor CoAP estão mostradas entre as linhas 17 e 25. Elas respondem a requisições `GET` no recurso `agri` com informações sobre agricultura, através da classe `AgriResource()`. O servidor é mantido em execução indefinidamente até que o programa seja encerrado. A programação assíncrona com `asyncio` permite que o servidor trate múltiplas requisições de forma eficiente.

```
1  import asyncio
2  import random
3  import struct
4  from aiocoap import *
5  from aiocoap.numbers.codes import Code
6  from aiocoap import resource
7
8  class AgriResource(resource.Resource):
9      async def render_get(self, request):
10         data = self.generate_sensor_data()
11         return Message(payload=data, code=Code.CONTENT, content_format
       =0)
12     def generate_sensor_data(self):
13         temperature = random.uniform(15.0, 35.0) #Temperatura em graus
       Celsius
14         humidity = random.uniform(20.0, 80.0)  # Umidade do solo em
       porcentagem
15         sensor_data = struct.pack('ff', temperature, humidity)
16         return sensor_data + bytes(500 - len(sensor_data))
17 async def main():
18     # Configura o servidor CoAP
19     root = resource.Site()
20     root.add_resource(['agri'], AgriResource())
21     await Context.create_server_context(root)
22     print("Servidor CoAP de Agricultura iniciado...")
23     await asyncio.get_running_loop().create_future()  # Mantem o
       servidor rodando
24 if __name__ == "__main__":
25     asyncio.run(main())
```

Código 4.2 - Exemplo de servidor CoAP.

MQTT

O *broker* `mosquitto` e a biblioteca `paho-mqtt` são utilizados na implementação do protocolo MQTT. Este protocolo necessita o desenvolvimento de dois tipos de clientes, os subscritores e os publicadores. Os subscritores e os publicadores se comunicam através do *broker* `mosquitto`.

O Código 4.3 mostra um exemplo de subscritor MQTT. O subscritor se conecta ao *broker* com endereço IP 192.168.1.3, através da porta 1883, e com tópico `agriculture/data`. As portas 1883 e 8883 são portas padrão para conexões MQTT sem TLS/SSL e seguras com TLS/SSL, respectivamente.

O subscritor possui duas funções de `callback`. A primeira função de `callback` recebe mensagens, e está definida entre as linhas 8 e 10. A segunda função de `callback`, linhas 13 a 15, conecta o cliente ao *broker*.

Um novo cliente é criado na linha 18. Este cliente é conectado ao *broker* através

da porta definida na linha 4.

```
1  import paho.mqtt.client as mqtt
2  import json
3  broker = '192.168.1.3'
4  port = 1883
5  topic = 'agriculture/data'
6
7  # Callback para mensagem recebida
8  def on_message(client, userdata, message):
9      data = json.loads(message.payload.decode())
10     print(f"Dados recebidos: {data}")
11
12 # Callback para quando o cliente se conecta ao broker
13 def on_connect(client, userdata, flags, rc, properties=None):
14     print(f"Conectado ao broker com codigo de resultado: {rc}")
15     client.subscribe(topic)
16
17 # Criando um novo cliente MQTT
18 client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
19 # Associando callback para quando se conectar
20 client.on_connect = on_connect
21 # Associando a callback para mensagem recebida
22 client.on_message = on_message
23 # Conectando ao broker
24 client.connect(broker, port)
25 # Mantendo o cliente em loop para ouvir mensagens
26 client.loop_forever()
```

Código 4.3 - Exemplo de subscritor MQTT.

A implementação de publicadores MQTT, também, exige o uso das configurações de *broker*, porta e tópico para estabelecer a conexão. No Código 4.4, estas configurações são defindas nas linhas 14, 15 e 16.

O publicador é responsável por transmitir dados de temperatura, em graus celsius para os sensores de temperatura, e umidade do solo, em porcentagem. Estes dados são gerados aleatoriamente pelas funções `generate-temperature()`, linhas 6 e 7, e `generate-humidity()`, linhas 9 e 10. Os dados de temperatura e umidade gerados perfazem um total de 50 bytes. Assumiu-se no exemplo, mensagens de 500 ou 5000 bytes. Então, é necessário gerar dados extras para completar os 500 ou 5000 bytes. Os dados extras são gerados pela função `generate-extra-data()`, que está definida nas linhas 12 e 13. Das linhas 18 a 32 podem ser vistos a criação do cliente, a estrutura dos dados transmitidos e sua publicação. Após a publicação, o publicador é desconectado usando a função `client.disconect()` definida na linha 34.

```
1  import paho.mqtt.client as mqtt
2  import json
```

```
import random
import string

def generate_temperature():
    return round(random.uniform(15.0, 35.0), 2)

def generate_humidity():
    return round(random.uniform(30.0, 90.0), 2)

def generate_extra_data(size):
    return ''.join(random.choices(string.ascii_letters + string.digits,
    k=size))
broker = '192.168.1.3'
port = 1883
topic = 'agriculture/data'

# Criando um novo cliente MQTT
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
# Conectando ao broker
client.connect(broker, port)
# Gerando os dados de temperatura e umidade
data = {
    'temperature': generate_temperature(),
    'humidity': generate_humidity(),
    'extra_data': generate_extra_data(450)  # Ajuste do valor para que o
    total seja aproximadamente 500 bytes
}
# Convertendo os dados para uma string JSON
data_json = json.dumps(data)
# Publicando os dados
client.publish(topic, data_json)
print(f"Dados enviados: {data_json}")
# Desconectando do broker
client.disconnect()
```

Código 4.4 - Exemplo de publicador MQTT.

Radnet Protocol - RP

A implementação de aplicações para experimentos no RP exigiu a definição de interesses, geração de prefixos de nós assim como seus identificadores. Para o encaminhamento de mensagens, também, é necessário definir nós vizinhos para cada nó. Foi usado o modelo cliente-servidor. O Código 4.5 ilustra a implementação de uma aplicação do lado servidor.

```

#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <ifaddrs.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <time.h>

#include "repa.h"

#define SYSLOGPORT 7514
#define INTEREST_TEMPERATURE_HUMIDITY "temperature-humidity"

struct info_socket {
    repa_sock_t socket;
    char interest[50];
} sock1;

void error(int no, const char *msg) {
    if (no < 0) {
        fprintf(stderr, "Error: %s (No: %d \"%s\").\n", msg, no,
    strerror(no));
        exit(EXIT_FAILURE);
    }
}

char* get_container_ip() {
    struct ifaddrs *ifaddr, *ifa;
    static char ip[INET_ADDRSTRLEN];

    if (getifaddrs(&ifaddr) == -1) {
        perror("getifaddrs");
        exit(EXIT_FAILURE);
    }

    for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr == NULL) continue;
        if (ifa->ifa_addr->sa_family == AF_INET) {
            struct sockaddr_in *sa = (struct sockaddr_in *)ifa->ifa_addr
    ;
            inet_ntop(AF_INET, &(sa->sin_addr), ip, INET_ADDRSTRLEN);
            if (strcmp(ip, "127.0.0.1") != 0) {
                freeifaddrs(ifaddr);
                return ip;
            }
        }
```

```
49      }
50      freeifaddrs(ifaddr);
51      return NULL;
52  }
53
54  void print_socket_info(int sockfd) {
55      struct sockaddr_in addr;
56      socklen_t addr_len = sizeof(addr);
57      if (getsockname(sockfd, (struct sockaddr *)&addr, &addr_len) == -1)
        {
58          perror("getsockname");
59          return;
60      }
61      char ip_str[INET_ADDRSTRLEN];
62      inet_ntop(AF_INET, &(addr.sin_addr), ip_str, INET_ADDRSTRLEN);
63      printf("Servidor registrado no IP: %s e porta: %d\n", ip_str, ntohs(
        addr.sin_port));
64  }
65
66  int main() {
67      srand(time(NULL));  // Inicializa o gerador de numeros aleatorios
68      int sockfd;
69      struct sockaddr_in servaddr;
70
71      sockfd = socket(AF_INET, SOCK_DGRAM, 0);
72      if (sockfd < 0) {
73          perror("socket");
74          exit(EXIT_FAILURE);
75      }
76
77      char* container_ip = get_container_ip();
78      if (container_ip == NULL) {
79          fprintf(stderr, "Nao foi possivel obter o IP do container.\n");
80          exit(EXIT_FAILURE);
81      }
82
83      bzero(&servaddr, sizeof(servaddr));
84      servaddr.sin_family = AF_INET;
85      servaddr.sin_addr.s_addr = inet_addr(container_ip);
86      servaddr.sin_port = htons(SYSLOGPORT);  // Porta 7514
87
88      if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) ==
        -1) {
89          perror("bind");
90          exit(1);
91      }
92
```

```
93      print_socket_info(sockfd);
94
95      prefix_addr_t prefix_addr;
96      char cprefix[20];
97
98      sock1.socket = repa_open();
99      error(sock1.socket.error, "abrir repa");
100
101     strcpy(sock1.interest, INTEREST_TEMPERATURE_HUMIDITY);
102     repa_register_interest(&sock1.socket, sock1.interest);
103
104     prefix_addr = repa_get_node_address();
105     repa_print_prefix(prefix_addr, cprefix);
106     printf("Servidor RADNET Prefixo: %u %s\n", prefix_addr, cprefix);
107
108     char data[LIBREPA_MAX_DATA_LEN(700)];
109
110     while (1) {
111         // Simulando dados de temperatura e umidade
112         double temperature = (rand() % 100);  // Gera um numero
    aleatorio para temperatura
113         double humidity = (rand() % 100);      // Gera um numero
    aleatorio para umidade
114         sprintf(data, "Temperature: %.2f, Humidity: %.2f", temperature,
    humidity);
115
116         // Preencher com espacos ate que os dados tenham 500 bytes
117         size_t data_len = strlen(data);
118         if (data_len < 500){
119             memset(data + data_len, ' ', 500 - data_len);//preenche com
    espacos
120         }
121         data[500 - 1] ='\0';// Garante que a string seja terminada
    corretamente
122
123         printf("Enviando dados de temperatura e umidade: %s\n", data);
124         repa_send(&sock1.socket, INTEREST_TEMPERATURE_HUMIDITY, data,
    500, 0);
125         sleep(2);  // Espera 2 segundos antes de enviar novos dados
126     }
127
128     repa_close(&sock1.socket);
129     close(sockfd);
130     return 0;
```

```
131 }
```

Código 4.5 - Exemplo de implementação de aplicação do lado servidor para mensagens de 500 Bytes.

O Código 4.5 implementa uma aplicação, utilizando o protocolo RadNet, projetada para coletar e transmitir dados de temperatura e umidade em intervalos regulares. Essa aplicação é composta por diversas funções que facilitam a comunicação via *sockets* e a integração com o protocolo RadNet.

No início do código, são incluídas as bibliotecas necessárias, como `<stdio.h>`, `<stdlib.h>`, e `<arpa/inet.h>`, que fornecem as funcionalidades essenciais para a criação de *sockets* e manipulação de endereços IP. A aplicação também define a constante `SYSLOGPORT`, que especifica a porta utilizada para a comunicação (7514) e a *string* de interesse `INTEREST_TEMPERATURE_HUMIDITY`, que indica os dados que o servidor registra e envia.

Função `error()` é responsável por tratar erros que possam ocorrer durante a execução do programa. Caso um erro ocorra uma mensagem é exibida e o programa é encerrado.

A função `get_container_ip()` obtém o endereço IP do *container* em que a aplicação está sendo executada. Utilizando a função `getifaddrs()`, ela percorre as interfaces de rede disponíveis e retorna o primeiro IP que não seja o *localhost* (127.0.0.1).

A função `print_socket_info` é utilizada para exibir o endereço IP e a porta do *socket* que foi vinculado à aplicação, permitindo uma verificação visual da configuração de rede.

A função principal `main()` inicia a aplicação, criando um *socket* UDP e vinculando-o ao IP e à porta definidos. O uso da função `rand()` simula a geração de dados de temperatura e umidade. O servidor gera valores aleatórios para temperatura e umidade.

A *string* de dados é preenchida com espaços até alcançar 500 bytes, garantindo que a estrutura de dados enviada seja consistente com as expectativas do protocolo. Finalmente, função `main()` envia os dados utilizando a função `repa_send()`, que é uma chamada específica para o protocolo RadNet.

O Código 4.5 implementa uma aplicação, utilizando o protocolo RadNet, projetada para coletar e transmitir dados de temperatura e umidade em intervalos regulares. Essa aplicação é composta por diversas funções que facilitam a comunicação via *sockets* e a integração com o protocolo RadNet. Além deste servidor, também foram desenvolvidas aplicações do lado cliente, que se conectam ao servidor para receber e processar os dados transmitidos. O Código 4.6 representa um exemplo de aplicação do lado cliente.

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
```

```c
#include <unistd.h>
#include <arpa/inet.h>
#include <ifaddrs.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include "repa.h"

#define SYSLOGPORT 7514
#define BUFFER_SIZE 1024

char* get_container_ip() {
    struct ifaddrs *ifaddr, *ifa;
    static char ip[INET_ADDRSTRLEN];

    if (getifaddrs(&ifaddr) == -1) {
        perror("getifaddrs");
        exit(EXIT_FAILURE);
    }

    for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr == NULL) continue;
        if (ifa->ifa_addr->sa_family == AF_INET) {
            struct sockaddr_in *sa = (struct sockaddr_in *)ifa->ifa_addr
    ;
            inet_ntop(AF_INET, &(sa->sin_addr), ip, INET_ADDRSTRLEN);
            if (strcmp(ip, "127.0.0.1") != 0) {
                freeifaddrs(ifaddr);
                return ip;
            }
        }
    }
    freeifaddrs(ifaddr);
    return NULL;
}

int main() {
    repa_sock_t sock1;
    char interest[] = "temperature-humidity";
    char data[BUFFER_SIZE];
    prefix_addr_t src_address;
    prefix_addr_t dst_address;

    // Abrir o socket
    sock1 = repa_open();
    if (sock1.sock.sock_raw < 0) {
        fprintf(stderr, "Erro ao abrir socket: %d\n", sock1.error);
```

```
51          return EXIT_FAILURE;
52      }
53
54      // Registrar o interesse
55      if (repa_register_interest(&sock1, interest) != OK) {
56          fprintf(stderr, "Erro ao registrar interesse: %d\n", sock1.error
    );
57          repa_close(&sock1);
58          return EXIT_FAILURE;
59      }
60
61      // Imprimir o IP e a porta do cliente
62      char ip_str[INET_ADDRSTRLEN];
63      struct sockaddr_in servaddr;
64      servaddr.sin_family = AF_INET;
65      servaddr.sin_port = htons(SYSLOGPORT);
66      servaddr.sin_addr.s_addr = inet_addr(get_container_ip());
67
68      inet_ntop(AF_INET, &(servaddr.sin_addr), ip_str, INET_ADDRSTRLEN);
69      printf("Cliente registrado no IP: %s e porta: %d\n", ip_str,
    SYSLOGPORT);
70
71      // Receber dados
72      while (1) {
73          int bytes_received = repa_recv(&sock1, interest, data, sizeof(
    data), &src_address, &dst_address);
74          if (bytes_received < 0) {
75              fprintf(stderr, "Erro ao receber dados: %d\n",
    bytes_received);
76          } else {
77              data[bytes_received] = '\0'; // Garantir que os dados sejam
    uma string
78              printf("Dados recebidos: %s\n", data);
79              printf("Endereco de origem: %u\n", src_address);
80              printf("Endereco de destino: %u\n", dst_address);
81          }
82
83          sleep(1); // Espera um segundo antes de tentar receber novos
    dados
84      }
85
86      // Desregistrar o interesse e fechar o socket
87      repa_unregister_interest(&sock1, interest);
88      repa_close(&sock1);
89
90      return EXIT_SUCCESS;
91  }
```

```
92
93 }
```

Código 4.6 - Exemplo de implementação de aplicação do lado cliente

4.8 Experimentos Realizados

As ferramentas utilizadas nos experimentos para coleta e análise de dados são: **tcpdump, wireshark, iperf** e **iproute2**. O **tcpdump** foi usado na captura de pacotes. Posteriormente, os dados são analisados com o **wireshark**, porque ele por si só não captura pacotes que trafegam em *namespaces*. O **iperf** e **iproute2** são usados no gerenciamento de largura de banda nas redes de *namespaces*.

Os experimentos consideram cenários de redes de 3, 5 ou 10 nós para os protocolos em estudo CoAP, MQTT e Radnet Protocol, conforme descrito anteriormente.

Cada experimento consistiu em 30 requisições ao servidor e/ou *broker*. Para cada uma das requisições o servidor envia ao cliente uma mensagem de confirmação de recebimento da mensagem do cliente. Desta forma, o cliente pode proceder o envio da próxima mensagem. A carga útil das mensagens contém dados de temperatura e umidade gerados pelas aplicações, simulando, desta forma, os dados de sensores de temperatura e umidade do solo. Para simular mensagens de 500 bytes ou 5000 bytes, os dados de temperatura e umidade são complementados com dados extras gerados aleatoriamente, que formam a carga útil necessária nos dois casos.

Resumindo, os experimentos são composto por três dimensões combinadas:

1. Três (3) protocolos: CoAP, MQTT e RadNet;
2. Quatro (4) métricas quantitativas: RTT, latência, *jitter* e taxa de entrega de pacotes;
3. Três (3) métricas de escalabilidade: tamanho de mensagens, largura de banda e número de dispositivos.

Em todas as combinações de experimentos geradas são assumidas as seguintes características de ambiente:

1. Número de dispositivos: 3, 5 ou 10 dispositivos;
2. Largura de banda: 30 ou 50Kbps;
3. Tamanho de mensagens: 500 ou 5000 Bytes.

5 ANÁLISE DOS RESULTADOS

Neste capítulo são apresentados e discutidos os resultados obtidos nos experimentos definidos no Capítulo 4. A comparação dos resultados está separada por métricas e por protocolos. Para os protocolos COAP e MQTT são apresentados os resultados do RTT, latência, taxa de entrega e *jitter*, considerando largura de banda de 30 e 50 kbps e mensagens de 500 e 5000 bytes. Para o protocolo RadNet é apresentado o *jitter* a 30 e 50 kbps e mensagens de 50 bytes.

Foi realizado o tratamento estatístico com base no número de amostras coletadas. Considerou-se o valor médio das amostras coletadas, o desvio padrão e o intervalo de confiança de 95% para as amostras. Para cada experimento foram coletadas 30 medições na comunicação entre clientes e servidor/*broker* de forma concorrente.

Em todos os experimentos não foi considerada a camada de segurança para todos protocolos. Para o MQTT definiu-se *QoS* de Nível 2.

O RTT, a latência e o *jitter* estão apresentados em unidade de milissegundos (ms).

Para determinar a latência, foi assumido que a rede é simétrica, ou seja, o tempo de ida é aproximadamente igual ao tempo de volta dos pacotes. Portando, o RTT foi dividido por 2, para obter a latência de ida, isto é: $Latencia = RTT/2$. No ambiente de *namespaces* essa suposição é particularmente válida, uma vez que os *namespaces* compartilham a mesma máquina física, garantindo rotas simétricas e um controle mais rígido das condições de tráfego. Como os *namespaces* estão isolados apenas logicamente, mas operam sob o mesmo núcleo e infraestrutura de rede, a latência de ida e volta tende a ser muito próxima, sem variações significativas de rotas ou congestionamento.

Nas próximas seções são apresentadas tabelas que mostram o RTT, a latência e o *jitter* dos protocolos em estudos.

5.1 Round Trip Time

Esta seção apresenta os resultados de RTT combinados com a escalabilidade dos experimentos para cenários com largura de banda de 30 e 50 kbps, mensagens de 500 e 5000 bytes e o número de dispositivos de 3, 5 e 10 nós.

CoAP

A Tabela 5 mostra o RTT para o protocolo CoAP em milissegundos. Ela é composta por cinco colunas, onde a primeira indica o número de nós e as colunas restantes indicam a largura de banda e o tamanho de mensagens.

Tabela 5 - RTT do CoAP

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,534	244,353	0,518	134,136
5	0,566	242,877	0,512	116,883
10	0,511	233,435	0,594	133,616

Pode ser observado, a partir dos dados da tabela, que a largura de banda maior (50kbps) geralmente apresenta RTTs menores para pacotes maiores (5000 Bytes), sugerindo que uma maior largura de banda pode suportar transferências de dados de forma mais eficiente, especialmente para pacotes grandes. A diferença no RTT é menos significativa para pacotes pequenos (500 Bytes), indicando que a largura de banda tem um impacto maior no RTT quando o tamanho dos pacotes é grande.

A partir dos dados apresentados, pode-se notar também que, para a largura de banda de 30kbps, o RTT para 500 Bytes e 5000 Bytes não varia significativamente com o aumento do número de nós, sugerindo uma estabilidade relativa na rede. Para a largura de banda de 50 kbps, há uma leve variação no RTT para pacotes de 500 Bytes com o aumento do número de nós, mas para 5000 Bytes, a variação é mínima. Portanto, o aumento do número de nós na rede parece não afetar significativamente o RTT, indicando que a rede é escalável e pode suportar um maior número de dispositivos sem comprometer a latência de forma significativa.

Observando sob ponto de vista do tamanho dos pacotes, pode ser percebido que tanto para largura de banda de 30 kbps quanto para a largura de banda de 50 kbps, o RTT para pacotes de 5000 Bytes é significativamente maior do que para pacotes de 500 Bytes. Portanto, essa constatação sugere que o tamanho dos pacotes tem um impacto direto no RTT, com pacotes maiores apresentando maior RTT. Este fato é esperado, pois pacotes maiores demoram mais para ser transmitidos através da rede.

O aumento do número de nós aumenta a carga na rede, resultando em maior tempo de resposta. Redes com maior largura de banda (50 kbps) lidam melhor com o aumento do número de nós, mas ainda assim, o RTT aumenta com mais nós.

Vale ressaltar que para cenário de 5 nós a uma largura de banda de 50 kbps, o RTT para mensagens de 5000 bytes é menor, o mesmo apresenta valores muito similares para cenários com 3 e 10 nós. O comportamento otimizado com 5 nós provavelmente é resultado de eficiência de uso da largura de banda e menor contenção. A rede com 5 nós pode estar utilizando a capacidade de 50 kbps de forma mais eficiente, sem as ineficiências observadas com 3 nós (subutilização) ou com 10 nós (maior competição pelo meio).

MQTT

Na Tabela 6 pode-se observar o RTT para o protocolo MQTT. A tabela é composta por cinco colunas, onde a primeira indica o número de nós e as colunas restantes indicam a largura de banda e o tamanho de mensagens.

Tabela 6 - RTT do MQTT

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,149	557,185	0,897	349,441
5	2,871	745,411	0,683	440,140
10	387,513	883,589	213,16	634,612

Na Tabela 6 pode-se observar que o RTT é relativamente baixo para mensagens de 500 bytes e rede de três nós, mas aumenta consideravelmente para mensagens de 5000 bytes. Para rede de cinco nós, o RTT aumenta significativamente em relação à rede com três nós, especialmente para mensagens menores. No cenário de dez nós, o RTT cresce drasticamente, especialmente para mensagens menores, se comparado às redes de três e cinco nós.

Para rede de três nós e largura de banda de 30 Kbps, o RTT é ligeiramente maior para mensagens de 500 bytes do que para mensagens de 5000 bytes. Para o cenário de cinco nós e tamanho de mensagens de 500 bytes, o RTT é maior para largura de banda de 50 kbps. O cenário de dez nós mostra que o RTT continua a aumentar à medida que o número de nós cresce, mas é consideravelmente menor no cenário de 30 kbps.

Esses dados revelam algumas tendências importantes sobre o comportamento do RTT do protocolo MQTT em diferentes condições de largura de banda e tamanho de rede.

1. Impacto da largura de banda: a largura de banda de 50 kbps resulta em RTTs mais baixos em comparação com 30 kbps, especialmente para redes maiores, 5 e 10 nós. Isso é mais evidente em mensagens de 5000 bytes, onde o RTT em 50 kbps é significativamente menor. O aumento da largura de banda reduz a latência, mas o impacto é mais pronunciado à medida que o número de nós cresce e o tamanho das mensagens aumenta.

2. Impacto do número de nós: à medida que o número de nós aumenta, o RTT tende a crescer, independentemente da largura de banda. Este efeito é especialmente acentuado em redes com 10 nós. Em redes pequenas, 3 nós, a diferença de RTT entre 30 kbps e 50 kbps não é tão significativa, mas se torna mais evidente à medida que a rede cresce.

3. Impacto do tamanho de mensagens: mensagens maiores (5000 bytes) resultam em RTTs muito mais altos do que mensagens menores (500 bytes), o que é esperado devido ao tempo adicional necessário para transmitir e processar mais dados. A diferença no RTT para mensagens de diferentes tamanhos é mais pronunciada em cenários de baixa largura de banda e redes maiores.

Alguns pontos importantes devem ser destacados. Resultados apresentados para mensagens de 500 bytes e cenários de três nós, o RTT para largura de banda de 30 kbps (0,149) é menor que o RTT para largura de banda de 50 kbps (0,897). Nos cenários de cinco nós, também, houve esta constatação. O RTT para 30 kbps (2,871) é maior que o RTT para 50 kbps (0,683). Isto indica que o comportamento do protocolo MQTT é influenciado por mais fatores do que apenas a largura da banda. A explicação para estas observeções estão a seguir.

Caso de 3 Nós (500 bytes)

No caso de três nós com 50 kbps, o aumento da largura de banda pode ter criado uma situação de subutilização da capacidade disponível. Com apenas três nós, a rede não gera tráfego suficiente para saturar os 50 kbps. Isso pode levar a maior latência de processamento ou de enfileiramento, pois o sistema de enfileiramento de pacotes pode funcionar de forma menos eficiente quando a rede está operando bem abaixo da capacidade máxima.

Além do exposto anteriormente, o MQTT pode ter apresentado *overhead* de controle maior com 50 kbps, já que a largura de banda maior permite que mais pacotes de controle (como pacotes de *handshake*, sincronização entre outros) sejam enviados para garantir a entrega dos pacotes. Isso pode aumentar o RTT, especialmente em redes menores com menos tráfego de dados útil.

Em redes com 30 kbps, pode haver menos necessidade de fragmentar pacotes ou retransmiti-los devido à menor competição pelo meio. O RTT mais baixo com 30 kbps pode ser resultado de uma transmissão mais eficiente de pacotes pequenos, já que a rede pode estar operando de forma mais estável e previsível com menos tráfego de controle ou colisões.

Caso de 5 Nós (500 bytes)

Quando a rede tem cinco nós com 30 kbps, o RTT aumenta significativamente (2,871). Isso sugere que a rede está enfrentando contenção pelo meio. O número de nós competindo para enviar pacotes pode gerar colisões ou necessidade de retransmissões, o que aumenta o RTT. A baixa largura de banda amplifica esse efeito, pois o tempo necessário para resolver essas colisões ou reenviar pacotes é maior.

Além disto, com cinco nós e 50 kbps, a rede tem mais largura de banda disponível, o que reduz a contenção pelo meio. A maior largura de banda permite que mais pacotes sejam enviados simultaneamente, resultando em um RTT menor (0,683). Isso sugere que a rede com cinco nós se beneficia de uma maior largura de banda, aproveitando melhor a

capacidade extra para evitar congestionamento e reduzir o tempo de espera.

O MQTT pode operar de forma mais eficiente com cinco nós e 50 kbps, lidando melhor com a maior carga de tráfego. Isso pode ser devido a melhores mecanismos de controle de congestionamento, como menos colisões, menor número de retransmissões e melhor gerenciamento de filas de pacotes. Como resultado, o RTT com 50 kbps e cinco nós é significativamente menor do que em 30 kbps.

De forma resumida podemos afirmar que, o comportamento observado é um reflexo da interação entre o número de nós, o tamanho dos pacotes e a largura de banda disponível. Redes pequenas com alta largura de banda podem ter problemas de subutilização, enquanto redes maiores podem se beneficiar de largura de banda extra para mitigar problemas de congestionamento e colisões.

5.2 Latência

Nesta seção são apresentados resultados para latência, combinados com a escalabilidade dos experimentos para cenários com largura de banda de 30 e 50 kbps, mensagens de 500 e 5000 bytes e o número de dispositivos de 3, 5 e 10 nós.

CoAP

A latência para o protocolo CoAP é apresentada na Tabela 7. Observando os dados da tabela sob o ponto de vista do número de nós, para pacotes de 500 Bytes e largura de banda de 30 kbps a latência apresenta uma leve variação, mas permanece estável à medida que o número de nós aumenta. Para o mesmo tamanho de pacotes e largura de banda de 50 kbps, similarmente, a latência permaneceu quase constante, com uma leve tendência de aumento ao se atingir 10 nós.

Tabela 7 - Latência do CoAP

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,267	122,177	0,259	67,068
5	2,283	121,439	0,256	58,442
10	0,256	116,718	0,297	66,808

Considerando pacotes de 5000 Bytes, quando a largura de banda é de 30 kbps, observa-se uma redução na latência à medida que o número de nós aumenta. Este comportamento pode ser atribuído à maior eficiência no uso da largura de banda disponível, possivelmente devido a melhor distribuição de carga entre os nós. Para largura de banda de 50 kbps a latência é significativamente menor para 5 nós, sugerindo que este é um ponto de eficiência ideal. A latência aumenta ligeiramente com 10 nós, indicando que a sobre-

carga de coordenação entre um número maior de nós começa a impactar negativamente a performance.

Fazendo uma avaliação em relação à largura de banda, para pacotes de 500 Bytes, a diferença na latência entre 30 kbps e 50 kbps é mínima, indicando que pacotes menores não são tão sensíveis à variação de largura de banda dentro do estudo feito. Para 5000 Bytes a latência é significativamente reduzida com 50 kbps em comparação com 30 kbps. Este resultado é esperado, já que pacotes maiores se beneficiam mais de uma largura de banda maior, resultando em tempos de transmissão mais curtos e, consequentemente, menor latência.

MQTT

Na Tabela 8 são apresentados os resultados da latência para 30 kbps e 50 kbps. Pode-se observar que, em cenário de três nós a latência é baixa para mensagens de 500 bytes, mas aumenta significativamente para mensagens de 5000 bytes. A latência aumenta à medida que o número de nós aumenta assim como quando aumenta o tamanho de mensagens. Mas esse aumento é muito acentuado para cenário de dez nós.

Tabela 8 - Latência do MQTT

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,075	278,593	0,449	174,721
5	1,436	372,706	0,342	220,070
10	193,757	441,795	106,580	317,306

Sob a perspectiva de largura de banda, pode-se observar que para o cenário de três nós há uma redução quando é aplicada uma largura de banda de 30 kbps em mensagens maiores. Também pode ser observada uma diminuição de latência para cinco nós. No cenário de dez nós pode-se observar que a latência continua a aumentar com o número de nós, mas significativamente menor em comparação com 30 kbps.

Estes dados oferecem *insights* importantes sobre o comportamento da latência em redes MQTT sob diferentes condições de largura de banda, número de nós e tamanho de mensagens, que estão destacados a seguir.

1. Impacto da largura de banda: a latência é menor em 50 kbps em comparação com 30 kbps, o que sugere que uma largura de banda maior melhora o tempo de entrega das mensagens, especialmente em cenários com mais nós e mensagens maiores. A diferença de latência entre as duas larguras de banda é mais pronunciada para mensagens de 5000 bytes, o que indica que a largura de banda mais alta é mais eficaz na redução da latência para dados maiores.

2. Impacto do número de nós: a latência aumenta com o número de nós, indepen-

dentemente da largura de banda. Isso é especialmente evidente em redes com 10 nós, onde a latência para mensagens de 500 bytes e 5000 bytes é consideravelmente maior. Em cenários com menos nós (3 ou 5), o impacto do aumento do número de nós na latência é menos evidente, mas ainda assim perceptível.

3. Impacto do tamanho da mensagem: mensagens maiores resultam em latências significativamente mais altas do que mensagens menores. Esse comportamento é consistente em ambos os cenários de largura de banda. A diferença na latência entre mensagens de tamanhos diferentes é mais acentuada em redes de 10 nós, o que indica que redes maiores e mensagens maiores representam um desafio adicional para a eficiência do protocolo MQTT.

5.3 *Jitter*

Nesta seção são apresentados resultados para o *jitter*, combinados com a escalabilidade dos experimentos para cenários com largura de banda de 30 e 50 kbps, mensagens de 500 e 5000 bytes e o número de dispositivos de 3, 5 e 10 nós. Os dados de *jitter* estão organizados em tabelas com colunas que representam a largura de banda e o tamanho das mensagens, enquanto as linhas representam o número de nós na rede.

CoAP

A Tabela 9 apresenta resultados do *jitter* do protocolo CoAP. Pode-se observar o impacto da largura de banda, número de nós e tamnho de mensagens.

Tabela 9 - *Jitter* do CoAP.

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,040	29,910	0,019	30,133
5	0,038	31,685	0,022	31,861
10	0,019	32,531	0,055	31,377

Pode ser observado que para largura de banda de 30 kbps e mensagens pequenas (500 bytes), o *jitter* é relativamente baixo (entre 0.019 e 0.04) e tende a reduzir ligeiramente com o aumento de nós. Na mesma largura de banda de 30 kbps e mensagens grandes (5000 bytes), o *jitter* aumenta substancialmente (29.91 a 32.531), mostrando uma tendência de aumento com o número de nós.

Para largura de banda de 50 kbps, mensagens pequenas (500 bytes) apresentam *jitter* baixo, mas levemente crescente com mais nós (de 0.019 a 0.055). Nota-se que para mesma largura de banda de 50 kbps e mensagens grandes (5000 bytes), o *jitter* permanece alto, mas não varia significativamente com o número de nós.

A explicação para estas constatações estão a seguir.

1. Impacto do tamanho de mensagens: as mensagens maiores (5000 bytes) geraram *jitter* muito mais alto do que as mensagens menores (500 bytes). Este efeito foi observado em ambas as larguras de banda (30 kbps e 50 kbps). Mensagens maiores exigem mais tempo para serem transmitidas, especialmente em redes com largura de banda limitada. Isso aumenta a variabilidade temporal (*jitter*) nas entregas das mensagens, especialmente em redes congestionadas. Isso sugere que o uso de mensagens menores pode ser benéfico em cenários onde é necessário minimizar o *jitter*, especialmente em aplicações que exigem latência baixa e previsível. O tamanho das mensagens no CoAP, portanto, deve ser configurado cuidadosamente com base nos requisitos da aplicação.

2. Impacto da largura de banda: embora a largura de banda mais alta (50 kbps) não elimine o *jitter* para mensagens grandes, ela ajuda a manter o *jitter* um pouco mais controlado em relação à banda de 30 kbps, especialmente para mensagens pequenas (500 bytes). A largura de banda maior permite ao protocolo transmitir pacotes com menos atraso, o que reduz as variações temporais em redes mais congestionadas. Isso sugere que aumentar a largura de banda pode ser uma solução parcial para mitigar o *jitter*, embora não resolva totalmente o problema para mensagens grandes. Este fator é relevante para dimensionar a rede de acordo com os requisitos de comunicação dos nós.

3. Impacto do número de nós: para mensagens menores (500 bytes), o aumento do número de nós teve um impacto leve no *jitter*, aumentando ligeiramente conforme mais nós se conectavam. Em contraste, para mensagens grandes, o *jitter* mostrou uma leve variação ou se manteve estável. A variação no *jitter* com o número de nós sugere que, enquanto o aumento de nós pode intensificar a disputa por recursos de rede, o impacto é menos significativo do que o tamanho da mensagem ou a largura de banda. Esse comportamento pode ser explicado pela capacidade do protocolo CoAP de lidar com concorrência moderada entre nós. No entanto, se o número de nós aumentar muito além de 10, o *jitter* pode se tornar mais pronunciado.

MQTT

A Tabela 10 apresenta resultados do *jitter* para o protocolo MQTT para todos cenários, tomando como base o número de nós, tamanho de mensagens e largura de banda.

Observando os dados da tabela, pode-se notar que conforme o número de nós aumenta, o *jitter* tende a variar. Isso geralmente ocorre devido ao aumento de tráfego e à maior quantidade de dados processados simultaneamente pela rede.

Tabela 10 - *Jitter* do MQTT

Nós	**30 kbps (500 Bytes)**	**30 kbps (5000 Bytes)**	**30 kbps (500 Bytes)**	**50 kbps (5000 Bytes)**
3	0,040	417,950	0,048	175,964
5	2,382	274,119	0,039	267,853
10	61,702	625,672	37,828	395,953

Analisando sob o ponto de vista do tamanho de mensagens e largura de banda pode se observar que para largura de banda de 30 Kbps com mensagens de 500 bytes, o *jitter* parece relativamente pequeno com poucos nós, mas aumenta à medida que o número de nós cresce. Na mesma banda com mensagens de 5000 bytes, o *jitter* é significativamente maior do que com mensagens menores, o que indica que o tamanho da mensagem impacta substancialmente a estabilidade da rede.

Quando a largura é de 50 Kbps com mensagens de 500 bytes, o *jitter* é mais baixo em comparação com a largura de banda de 30 Kbps, sugerindo que a largura de banda mais alta pode reduzir a variação de atraso. Nesta mesma banda para mensagens de 5000 bytes, observa-se que o *jitter* é ainda considerável, o que pode indicar que um tamanho maior de mensagem aumenta a variabilidade, apesar da maior largura de banda.

A seguir são apresentados alguns pontos de análise para aprofundar a interpretação dos resultados da tabela.

1. Impacto do número de nós: à medida que o número de nós na rede aumenta, o *jitter* também tende a aumentar. Este comportamento é esperado, pois mais nós geram maior concorrência para enviar e receber dados, sobrecarregando o canal de comunicação e introduzindo mais variabilidade nos tempos de entrega das mensagens. Esse efeito é especialmente acentuado em redes com largura de banda menor, pois a capacidade de transporte de dados é mais limitada.

2. Impacto da largura: comparando os valores de *jitter* para 30 Kbps e 50 Kbps nas mesmas condições de carga, percebe-se que uma largura de banda maior (50 Kbps) geralmente resulta em menor *jitter* para o mesmo número de nós e tamanho de mensagem. Isso sugere que redes com largura de banda mais ampla conseguem acomodar um volume maior de dados com menor variabilidade de atraso. Isso ocorre porque uma maior largura de banda permite que pacotes de dados transitem com menor risco de congestionamento, reduzindo os picos de atraso.

3. Impacto do tamanho de mensagens: observa-se que, para o mesmo número de nós e largura de banda, mensagens maiores (5000 bytes) introduzem mais *jitter* do que mensagens menores (500 bytes). Mensagens de maior tamanho exigem mais tempo para serem transmitidas e ocupam mais espaço na rede, o que aumenta as chances

de congestionamento e, portanto, maior variação nos atrasos de entrega.

RadNet

A Tabela 11 apresenta resultados do *jitter* para o protocolo RadNet. Como mencionado em capítulos anteriores, o protocolo RadNet, atualmente, suporta apenas mensagens menores, sendo os resultados apresentados na tabela referentes a mensagens de 500 bytes. Desta forma, a tabela é composta por colunas que indicam a largura de banda e as linhas que indicam o número de nós.

Tabela 11 - *Jitter* do RadNet.

Nós	**30 kbps**	**50 kbps**
3	390,79	0,126
5	84,279	734,199
10	514,128	520,143

À semelhança dos outros protocolos, os dados de *jitter* do protocolo RadNet são discutidos, a seguir, e abordados os aspectos relacionados ao número de nós e a largura de banda.

1. Impacto da largura de banda: em redes com 3 nós, o *jitter* foi significativamente menor em 50 kbps (0,126) comparado a 30 kbps (390,790). Com 5 nós, ocorreu o inverso, o *jitter* foi mais alto em 50 kbps (734,199) do que em 30 kbps (84,279). Com 10 nós, o *jitter* permaneceu elevado em ambas as bandas, embora ligeiramente maior em 50 kbps (520,143) comparado a 30 kbps (514,128). A largura de banda parece influenciar o *jitter* de forma diferente dependendo da densidade da rede. Em redes menores (3 nós), uma banda mais alta reduz o *jitter* drasticamente, mas em redes mais densas (5 e 10 nós), a largura de banda maior não necessariamente melhora o *jitter*. Esse aumento inesperado em 5 nós para 50 kbps pode estar relacionado ao congestionamento e à sobrecarga de processamento na rede.

2. Impacto do número de nós: em redes com largura de banda de 30 kbps, o *jitter* aumentou substancialmente ao passar de 5 para 10 nós, de 84,279 para 514,128. Em 50 kbps, houve um aumento drástico do *jitter* ao passar de 3 nós (0,126) para 5 nós (734,199), mas o valor reduziu levemente para 10 nós (520,143). Esses resultados indicam que o aumento do número de nós provoca um crescimento acentuado no *jitter*, especialmente com uma largura de banda limitada de 30 kbps. A largura de banda de 50 kbps parece ser mais vulnerável ao efeito de "pico"de *jitter* ao passar de uma rede pequena (3 nós) para uma rede intermediária (5 nós). Esse comportamento sugere que há um limite na quantidade de nós que podem coexistir eficientemente, além do qual a latência variável (*jitter*) se torna problemática devido ao congestionamento e à competição por recursos.

5.4 Taxa de entrega de pacotes

Em todos cenários de experimentos avaliados, o protocolo CoAP alcançou 100% da taxa de entrega de pacotes.

Para o caso do MQTT, a taxa de entrega foi de 100% em todos cenários com pacotes de 500 bytes, os resultados não foram satisfatórios para todos cenários de 5000 bytes. Nestes casos não foi possível atingir 100%, principalmente quando o número de nós aumenta. Este fato pode ser devido a necessidade de retransmissões e a necessidade de envio de vários pacotes de controle. Conforme mencionado em capítulos anteriores, o MTU padrão é de 1500 bytes, logicamente, pacotes de 5000 bytes precisam ser transmitidos mais de uma vez.

5.5 Comparação dos protocolos com base no RTT

Nesta seção é feita a comparação de dois protocolos de aplicação, o CoAP e o MQTT. Para facilitar a compreensão, são apresentados os resultados do RTT dos dois protocolos na Tabela 12.

Tabela 12 - Comparação dos protocolos CoAP e MQTT com base no RTT

Rtt - 30 kbps				
Nós	**CoAP**		**MQTT**	
	500 Bytes	**5000 Bytes**	**500 Bytes**	**5000 Bytes**
3	0,534	244,353	0,149	557,185
5	0,566	242,877	2,871	745,411
10	0,511	233,435	387,513	883,589
Rtt - 50 kbps				
Nós	**CoAP**		**MQTT**	
	500 Bytes	**5000 Bytes**	**500 Bytes**	**5000 Bytes**
3	0,518	134,136	0,897	349,441
5	0,512	116,883	0,683	440,140
10	0,594	133,616	213,16	634,612

Analisando os dados da Tabela 12, pode-se constatar que para tamanhos de mensagem menores (500 bytes), o CoAP apresentou valores de RTT consistentemente mais baixos em comparação ao MQTT em todas as configurações de nós e larguras de banda. Isso indica que o CoAP é mais eficiente para cargas úteis pequenas em ambientes restritos.

Para mensagens maiores (5000 bytes), ambos os protocolos apresentaram um aumento no RTT, mas o MQTT apresentou RTT significativamente mais alto do que o CoAP. Isso sugere que o MQTT enfrenta dificuldades com cargas maiores, provavelmente

devido à sobrecarga do TCP.

Os valores apresentados se justificam porque o CoAP foi projetado para operar sobre UDP, o que permite uma comunicação mais leve e rápida. Não há a necessidade de estabelecer uma conexão antes da troca de mensagens. Isso resulta em menor latência, especialmente para cargas úteis pequenas. Em contraste, o MQTT utiliza TCP, que requer uma conexão estável e pode introduzir latências adicionais devido ao processo de *handshake* e à garantia de entrega das mensagens.

Nota-se igualmente que, aumentar a largura de banda de 30 kbps para 50 kbps reduz o RTT para ambos os protocolos,. Porém, o CoAP mantém uma vantagem mais significativa sobre o MQTT em termos de latência. A diferença no desempenho se torna menos pronunciada à medida que o número de nós aumenta, indicando que o congestionamento da rede pode impactar ambos os protocolos de maneira semelhante ao escalar.

À medida que o número de nós aumenta, o CoAP mantém uma latência mais baixa para mensagens menores, enquanto o desempenho do MQTT diminui mais notavelmente com mensagens maiores. Isso pode ser crítico ao implantar soluções IoT onde muitos dispositivos se comunicam frequentemente com pacotes pequenos de dados.

De uma forma resumida, o CoAP superou o MQTT em termos de RTT para tamanhos menores de mensagem em vários cenários, especialmente em ambientes restritos com largura de banda limitada. No entanto, enquanto o MQTT pode ser vantajoso em cenários que exigem entrega confiável de mensagens, seu desempenho se deteriora com mensagens maiores e maior número de nós em comparação com o CoAP. Essas informações podem orientar decisões sobre a seleção do protocolo com base nos requisitos específicos da aplicação e nas condições da rede.

5.6 Comparação dos protocolos com base no *jitter*

Nesta secção é feita a comparação dos três protocolos com base no *jitter*. Pelo fato do protocolo RadNet não suportar mensagens maiores, os resultados apresentados a seguir são referentes a mensagens de 500 bytes para todos protocolos.

A Tabela 13 mostra os resultados dos três protocolos para a métrica *jitter*. Ela é composta por colunas que representam os protocolos nas diferentes larguras de banda e as linhas que representam o número de nós.

O CoAP apresenta baixo *jitter* tanto em 30 kbps quanto em 50 kbps, especialmente para redes de 3 a 5 nós, indicando estabilidade mesmo em condições de baixa largura de banda. O aumento do *jitter* com 10 nós, sobretudo em 50 kbps, pode sugerir alguma instabilidade em redes maiores, possivelmente devido a maior latência ou concorrência no canal.

Para o MQTT, no cenário de 3 nós, o *jitter* permanece moderadamente baixo para

Tabela 13 - Comparação dos protocolos CoAP, MQTT e RadNet com base no *jitter*.

Nós	**CoAP**		**MQTT**		**RadNet**	
	30 kbps	**50 kbps**	**30 kbps**	**50 kbps**	**30 kbps**	**50 kbps**
3	0,040	0,019	0,040	0,048	390,79	0,126
5	0,038	0,022	2,382	0,039	84,279	734,199
10	0,019	0,055	61,702	37,828	514,128	520,143

ambas as taxas de transmissão, com um leve aumento a 50 kbps. Ao expandir para 5 e 10 nós, o protocolo apresenta *jitter* muito mais alto, particularmente em 10 nós e a 30 kbps. Isso indica que, em redes maiores e com baixa largura de banda, o MQTT pode ser mais suscetível a congestionamentos e a quedas de desempenho, talvez devido à sobrecarga de mensagens de controle e ao gerenciamento de conexões.

No RadNet, no cenário de 3 nós, o *jitter* apresenta uma disparidade notável entre 30 kbps (muito alto, com 390 ms) e 50 kbps (0,126 ms), sugerindo que a largura de banda adicional ajuda a reduzir as variações. Com 5 nós, ocorre um aumento significativo do *jitter* para ambas as larguras de banda, e, em 10 nós, o *jitter* permanece alto. Isso sugere que o RadNet pode estar menos otimizado para redes maiores e talvez sensível à largura de banda disponível.

Comparando os três protocolos, pode-se afirmar que o CoAP apresenta um desempenho de *jitter* mais estável em redes de até 5 nós e tende a escalar melhor, especialmente sob taxa de 50 kbps. Isso sugere que ele pode ser mais adequado para cenários onde a consistência do atraso é crítica.

A escalabilidade da rede afeta significativamente o MQTT e o RadNet, com o *jitter* aumentando drasticamente. Essa característica é importante para considerar se a aplicação final é sensível a *jitter*, especialmente em redes IoT ou com mensagens em tempo real.

Sob o ponto de vista da largura de banda, em redes de 10 nós, o *jitter* do MQTT em 50 kbps é significativamente menor do que em 30 kbps. Isso pode indicar que, para redes maiores, o MQTT se beneficia mais de largura de banda adicional em comparação com o RadNet, onde o *jitter* permanece alto.

5.7 Discussão

Os resultados mostram que o RTT e latência do protocolo CoAP foram influenciados tanto pelo número de nós na rede quanto pela largura de banda disponível. A redução da latência com o aumento da largura de banda, especialmente para pacotes maiores, indica que redes com requisitos de baixa latência para grandes volumes de dados devem considerar a implementação de *links* de maior capacidade.

Por outro lado, a estabilidade da latência para pacotes menores sugere que, para certos tipos de aplicações onde os pacotes são tipicamente pequenos, aumentar a largura de banda pode não resultar em melhorias significativas. Isso é particularmente relevante para aplicações IoT que transmitem pequenos pacotes de dados sensoriais.

Além disso, a eficiência observada com um número intermediário de nós (5 nós) em alguns cenários indica que há um ponto de equilíbrio entre a quantidade de nós e o desempenho da rede. A sobrecarga de coordenação entre um grande número de nós pode começar a impactar negativamente a latência, mesmo com largura de banda maior.

Para o protocolo MQTT os resultados mostram que a largura de banda, o tamanho da rede e o tamanho das mensagens têm um impacto significativo no RTT. Redes maiores e mensagens maiores tendem a aumentar o RTT, especialmente em cenários de baixa largura de banda. Estes dados podem ser usados para otimizar o desempenho de redes MQTT, ajustando a largura de banda disponível ou limitando o tamanho das redes e das mensagens em aplicações críticas onde a latência é uma preocupação.

Os resultados também indicam que a latência em redes MQTT é fortemente influenciada pela largura de banda disponível, pelo número de nós na rede e pelo tamanho das mensagens. Redes com maior largura de banda apresentam latências menores, especialmente para mensagens maiores e em redes com mais nós. O aumento do número de nós e do tamanho das mensagens aumenta a latência, o que pode afetar o desempenho em aplicações onde a latência é crítica.

No que diz respeito ao *jitter*, em todos os cenários de número de nós e largura de banda, o CoAP apresentou um *jitter* relativamente estável e baixo, especialmente em redes de até 5 nós. Esse comportamento sugere que o CoAP é robusto em relação ao *jitter*, mantendo a consistência nas mensagens. Em uma rede de 10 nós, o *jitter* aumenta a 50 kbps, passando de 0,019 ms (em 3 nós) para 0,055 ms, o que, apesar de ser um aumento, ainda é consideravelmente baixo. Isso mostra que, enquanto o CoAP pode ser influenciado pelo tamanho de rede, ele permanece adequado para aplicações em redes maiores, como redes de sensores, onde a previsibilidade do atraso é importante.

O comportamento do MQTT revela uma sensibilidade maior ao tamanho da rede, com um aumento drástico no *jitter* à medida que o número de nós cresce. No cenário de 5 nós a 30 kbps, o *jitter* alcança 2,382 ms, e em uma rede de 10 nós o valor aumenta para 61,702 ms a 30 kbps e 37,828 ms a 50 kbps. Esse comportamento pode ser devido ao gerenciamento de conexões e à troca de mensagens de controle do MQTT, especialmente em uma arquitetura de publicação/subscrição onde cada nó publica e assina mensagens, sobrecarregando o sistema.

Esse aumento torna o MQTT mais desafiador para redes grandes e mais propenso a perdas de pacotes e reenvios, fatores que elevam o *jitter*. No entanto, em redes pequenas, o MQTT mantém um *jitter* controlado, o que o torna uma escolha útil para aplicações que requerem comunicação esporádica, como sistemas de monitoramento simples.

O RadNet apresenta um *jitter* extremamente elevado em redes maiores, com valores como 514 ms a 30 kbps em uma rede de 10 nós. Isso evidencia que o protocolo é sensível tanto ao tamanho da rede quanto à largura de banda disponível.

A alta variação do *jitter* entre 30 kbps e 50 kbps em 3 nós (de 390 ms a 0,126 ms) indica que o RadNet se beneficia muito de uma largura de banda maior, mas a tendência de aumento em redes com mais nós sugere que ele enfrenta dificuldades em ambientes com muitos nós, devido ao uso de comunicação por *broadcast*.

A comunicação por *broadcast* aumenta as chances de colisões e retransmissões de pacotes, particularmente em redes maiores, tornando o RadNet mais vulnerável ao congestionamento.

Fazendo uma análise comparativa com os trabalhos relacionados podemos elencar alguns aspectos importantes, destacados a seguir.

RTT

O CoAP apresentou um RTT menor, principalmente com pacotes menores (500 bytes) e largura de banda de 50 kbps. O protocolo se mostrou ser eficiente na transmissão de pacotes menores, independentemente do número de nós, e escalável, como indicado na Tabela 5.

Comparando com o trabalho de Silva *et al.* (SILVA et al., 2021), onde o CoAP também se destacou por ter a menor latência em relação ao MQTT, podemos notar uma consistência nos resultados de ambos os estudos.

O MQTT, por outro lado, mostrou um aumento significativo no RTT à medida que o número de nós e o tamanho dos pacotes aumentavam, principalmente em cenários de baixa largura de banda (30 kbps). A Tabela **??** mostra que o RTT chega a ser bastante elevado com 10 nós e pacotes de 5000 bytes. Isso está em concordância com o estudo de Bansal e Kumar (BANSAL; KUMAR, 2019), que indicam que o MQTT sofre mais com o aumento da latência em cenários de baixa largura de banda.

Latência

Para o CoAP, a latência se mostrou estável com pacotes menores e redes de até cinco nós, mas a eficiência diminuiu ligeiramente em redes maiores e pacotes de 5000 bytes, embora a largura de banda maior tenha ajudado a reduzir esse efeito. Isso reflete o que foi encontrado nos trabalhos relacionados, onde o CoAP se destaca pela baixa latência em comparação com outros protocolos de aplicação como o MQTT, conforme observado por Silva *et al.* (SILVA et al., 2021) e Bansal e Kumar (BANSAL; KUMAR, 2019).

A latência do MQTT aumentou significativamente com o número de nós e o tamanho dos pacotes. O comportamento do MQTT foi especialmente sensível à largura de banda, com melhor desempenho em 50 kbps do que em 30 kbps. O estudo de Ramirez e Pedraza (RAMIREZ; PEDRAZA, 2017) também observou que o aumento do número de nós e o tamanho dos pacotes resultam em uma latência mais alta para o MQTT, o que

está em concordância com os resultados observados.

Taxa de entrega

O CoAP teve 100% de taxa de entrega de pacotes em todos os cenários testados, mostrando uma robustez que também foi observada no estudo de Silva *et al.* (SILVA et al., 2021), onde o CoAP se destacou tanto em termos de latência quanto de perda de pacotes.

No MQTT os resultados foram menos satisfatórios, principalmente em redes maiores, onde tanto a latência quanto o RTT foram impactados. Isso é corroborado pelo estudo de Bansal e Kumar (BANSAL; KUMAR, 2019), que indicam que a latência no MQTT aumenta significativamente com a taxa de perda de pacotes.

CONSIDERAÇÕES FINAIS

Nesta pesquisa foi realizada uma análise de desempenho dos protocolos da camada de aplicação para IoT CoAP, MQTT e Radnet Protocol, em aplicações para agricultura. Existem várias bibliotecas disponíveis para a implementação de aplicações que usam estes protocolos, como `paho-mqtt`, `aiocoap` entre outras. Para avaliação dos protocolos foram implementadas aplicações para agricultura.

No desenvolvimento de novas aplicações para IoT pode se recorrer a simuladores, *testbeds* e emuladores para obter resultados de experimentação com o objetivo de evitar possíveis problemas que surgem ao se fazer uma implantação direta. Os experimentos de desempenho desta pesquisa foram realizados em diferentes redes de *namespaces* do ambiente Linux. Foi necessário variar a largura de banda, número de nós e o tamanho de pacotes enviados para um melhor estudo de cada protocolo em diferentes cenários de experimentos.

Com base nos resultados apresentados o protocolo CoAP mostrou-se superior em termos de latência, *jitter* e taxa de entrega, especialmente em redes menores e com pacotes menores. Ele se destaca pela robustez e escalabilidade, atingindo uma taxa de entrega de pacotes de 100% em todos os cenários testados. Estes resultados indicam que o CoAP apresenta uma boa escolha para aplicações IoT com requisitos de baixa latência e alta eficiência na entrega de pacotes, especialmente quando a largura de banda disponível é limitada.

Todos os protocolos tiveram seu desempenho diretamente influenciado pela largura de banda, com melhorias significativas em redes de 50 kbps. No entanto, o MQTT apresentou maior sensibilidade à largura de banda, especialmente em cenários com pacotes maiores. Isso sugere que o MQTT é menos eficiente em ambientes com baixa largura de banda, sendo mais adequado para cenários com largura de banda elevada, onde pode mitigar o aumento da latência.

O aumento do número de nós e o tamanho das mensagens impactaram negativamente o desempenho do MQTT, levando a aumentos significativos de latência e RTT. Em contraste, o CoAP apresentou maior estabilidade nestes aspectos, especialmente com pacotes menores, demonstrando que ele pode lidar melhor com redes maiores e volumes de tráfego moderados, sendo mais resiliente a variações na topologia da rede.

Portanto, o CoAP é recomendado para cenários IoT que envolvem transmissão de pequenos pacotes de dados com requisitos de baixa latência, como sensores ambientais e redes de dispositivos de baixo custo. O MQTT, por sua vez, pode ser mais adequado para cenários onde o tráfego de dados é maior e a largura de banda é suficiente para suportar a maior sobrecarga do protocolo.

O RadNet pode ser benéfico em cenários de transmissão única (*broadcast*) em redes

pequenas e com alta largura de banda, mas sua alta sensibilidade ao aumento de nós e ao uso de largura de banda mais baixa pode torná-lo uma escolha menos eficaz em redes densas, onde colisões e retransmissões podem impactar seriamente o desempenho.

O RadNet, embora menos eficaz em densidade elevada de nós, pode ser explorado em sistemas onde a largura de banda disponível é alta e o *jitter* não é um fator limitante crítico, desde que a rede não se expanda muito.

É importante ressaltar que, a escolha do melhor protocolo para uma aplicação IoT depende do compromisso entre o tamanho das mensagens, a largura de banda disponível, o número de dispositivos na rede e a necessidade de baixa latência. Protocolos como o CoAP oferecem maior estabilidade e eficiência em redes sensíveis à latência, enquanto o MQTT pode ser útil em cenários de maior largura de banda e onde a escalabilidade é menos crítica. Estas conclusões estão em conformidade com as observações de estudos anteriores, como os de Silva *et al.* (SILVA et al., 2021) e Bansal e Kumar (BANSAL; KUMAR, 2019), reforçando a importância de ajustar as configurações da rede às demandas específicas da aplicação.

Trabalhos futuros

Neste estudo, a camada de segurança foi desconsiderada. Um trabalho futuro poderia investigar o impacto de diferentes mecanismos de segurança (como DTLS para CoAP e TLS para MQTT) na latência, RTT, *jitter* e taxa de entrega, visto que a adição de segurança pode alterar significativamente o desempenho dos protocolos, especialmente em redes com recursos limitados.

No caso do MQTT, o QoS foi configurado no Nível 2. Trabalhos futuros poderiam explorar os impactos de outras configurações de QoS (Níveis 0 e 1) em termos de latência e perda de pacotes, avaliando se o *trade-off* entre confiabilidade e desempenho pode ser ajustado para diferentes tipos de aplicação IoT.

Atualmente o protocolo RadNet não suporta mensagens maiores, trabalhos futuros poderiam atualizar o protocolo de modo a suportar mensagens maiores.

Embora o estudo tenha incluído cenários com até 10 nós, seria interessante ampliar os experimentos para redes com dezenas ou centenas de dispositivos, o que representaria de forma mais precisa redes IoT massivas, como as utilizadas em cidades inteligentes. Além disso, o estudo poderia investigar como a escalabilidade dos protocolos é afetada em topologias de rede mais complexas.

O consumo de energia é um fator crítico em dispositivos IoT com recursos limitados, como sensores alimentados por baterias. Investigações futuras poderiam medir o impacto de cada protocolo no consumo de energia, avaliando o quanto o CoAP, MQTT e RadNet afetam a longevidade de dispositivos IoT em diferentes cenários de uso.

Muitos cenários IoT reais envolvem a coexistência de múltiplos protocolos de comunicação e tecnologias de rede (por exemplo, Wi-Fi, LoRa, 5G). Trabalhos futuros poderiam investigar como o CoAP, MQTT e RadNet se comportam em redes heterogêneas, onde diferentes dispositivos e protocolos interagem em um único ambiente.

REFERÊNCIAS

AHMAD, N.; LAPLANTE, P.; DEFRANCO, J. F. Life, iot, and the pursuit of happiness. *IT Professional*, IEEE, v. 22, n. 6, p. 4–7, 2020. 13

AIREEN, G. et al. Wireless network simulation and analysis using qualnet. In: IEEE. *2017 2nd international conference on communication and electronics systems (ICCES)*. [S.l.], 2017. p. 251–255. 45

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, IEEE, v. 17, n. 4, p. 2347–2376, 2015. 18, 19, 21, 22, 23, 24, 25, 28, 29, 30, 31, 32, 34, 35, 36, 38, 39, 40, 42

AL-ROUBAIEY, A.; AL-JAMIMI, H. Online power tossim simulator for wireless sensor networks. In: IEEE. *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. [S.l.], 2019. p. 1–5. 46

AL-SARAWI, S. et al. Internet of things (iot) communication protocols. In: IEEE. *2017 8th International conference on information technology (ICIT)*. [S.l.], 2017. p. 685–690. 36

ALHAIDARI, F. A.; ALQAHTANI, E. J. Securing communication between fog computing and iot using constrained application protocol (coap): A survey. *J. Commun.*, v. 15, n. 1, p. 14–30, 2020. 13, 21

ALMUHAYA, M. A. et al. A survey on lorawan technology: Recent trends, opportunities, simulation tools and future directions. *Electronics*, Multidisciplinary Digital Publishing Institute, v. 11, n. 1, p. 164, 2022. 43

AMQP Org. *AMQP*. 2023. Disponível em: <https://amqt.org>. Acesso em: 30 setembro 2023. 29

AMSüSS, C.; CONTRIBUTORS aiocoap. *aiocoap: Python CoAP Library*. 2013–. Disponível em: <https://christian.amsuess.com/tools/aiocoap/>. 58

BAKARE, B.; ENOCH, J. A review of simulation techniques for some wireless communication system. *International Journal of Electronics Communication and Computer Engineering*, v. 10, n. 2, p. 60–70, 2019. 50

BANSAL, S.; KUMAR, D. Iot application layer protocols: performance analysis and significance in smart city. In: IEEE. *2019 10th international conference on computing, communication and networking technologies (ICCCNT)*. [S.l.], 2019. p. 1–6. 51, 87, 88, 90

BARUA, A. et al. Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey. *IEEE Open Journal of the Communications Society*, IEEE, 2022. 37, 38

CAIZA, G. et al. Industrial shop-floor integration based on amqp protocol in an iot environment. In: IEEE. *2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM)*. [S.l.], 2019. p. 1–6. 30

CHAUDHARY, A.; PEDDOJU, S. K.; KADARLA, K. Study of internet-of-things messaging protocols used for exchanging data with external sources. In: IEEE. *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. [S.l.], 2017. p. 666–671. 30

CHEN, Y.; KUNZ, T. Performance evaluation of iot protocols under a constrained wireless access network. In: IEEE. *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*. [S.l.], 2016. p. 1–7. 31

CHERNYSHEV, M. et al. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, IEEE, v. 5, n. 3, p. 1637–1647, 2017. 47, 48, 49

CRUZ, M. A. da et al. A proposal for bridging the message queuing telemetry transport protocol to http on iot solutions. In: IEEE. *2018 3rd International Conference on Smart and Sustainable Technologies (SpliTech)*. [S.l.], 2018. p. 1–5. 33

DDS Foundtion Org. *What is DDS?* 2023. Disponível em: <https://www.dds-foundation.org/what-is-dds-3/>. Acesso em: 30 setembro 2023. 31

DENG, C. et al. Ieee 802.11 be wi-fi 7: New challenges and opportunities. *IEEE Communications Surveys & Tutorials*, IEEE, v. 22, n. 4, p. 2136–2166, 2020. 13, 19

DIAS, J. P. et al. A brief overview of existing tools for testing the internet-of-things. In: IEEE. *2018 IEEE international conference on software testing, verification and validation workshops (ICSTW)*. [S.l.], 2018. p. 104–109. 46, 48, 49

DUGAN, J. et al. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. 2003–. Disponível em: <https://iperf.fr/>. Acesso em: 30 Maio 2024. 60

DUTRA, R. C.; MORAES, H. F.; AMORIM, C. L. Interest-centric mobile ad hoc networks. In: IEEE. *2012 IEEE 11th International Symposium on Network Computing and Applications*. [S.l.], 2012. p. 130–138. 13, 27, 28, 51

Eclipse Foundation. *Paho*. 2021. Disponível em: <https://wiki.eclipse.org/Paho>. Acesso em: 30 Maio 2024. 58

ESFAHANI, A. et al. A lightweight authentication mechanism for m2m communications in industrial iot environment. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 1, p. 288–296, 2017. 34

FAROOQ, M. S.; AKRAM, S. Iot in agriculture: Challenges and opportunities. *Journal of Agricultural Research (03681157)*, v. 59, n. 1, 2021. 60

FAROOQ, M. S. et al. A survey on the role of iot in agriculture for the implementation of smart farming. *Ieee Access*, IEEE, v. 7, p. 156237–156271, 2019. 60

GARCIA-ESPINOSA, E. et al. Power consumption analysis of bluetooth low energy commercial products and their implications for iot applications. *Electronics*, MDPI, v. 7, n. 12, p. 386, 2018. 37

GHALEB, B. et al. A survey of limitations and enhancements of the ipv6 routing protocol for low-power and lossy networks: A focus on core operations. *IEEE Communications Surveys & Tutorials*, IEEE, v. 21, n. 2, p. 1607–1635, 2018. 35, 36

GHOTBOU, A.; KHANSARI, M. Comparing application layer protocols for video transmission in iot low power lossy networks: an analytic comparison. *Wireless Networks*, Springer, v. 27, p. 269–283, 2021. 34

GUPTA, B. B.; QUAMARA, M. An overview of internet of things (iot): Architectural aspects, challenges, and protocols. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 32, n. 21, p. e4946, 2020. 18, 19

GUPTA, H. et al. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, Wiley Online Library, v. 47, n. 9, p. 1275–1296, 2017. 44

GUPTA, P. et al. A survey of application layer protocols for internet of things. In: IEEE. *2021 International Conference on Communication information and Computing Technology (ICCICT)*. [S.l.], 2021. p. 1–6. 23, 25, 29

GYRARD, A.; SERRANO, M. Fiesta-iot: Federated interoperable semantic internet of things (iot) testbeds and applications. In: *ICT*. [S.l.: s.n.], 2015. 49

HOFER-SCHMITZ, K.; STOJANOVIĆ, B. Towards formal verification of iot protocols: A review. *Computer Networks*, Elsevier, v. 174, p. 107233, 2020. 21

JAIN, S.; CHOUDHARI, P.; SRIVASTAVA, A. The fundamentals of internet of things: architectures, enabling technologies, and applications. In: *Healthcare Paradigms in the Internet of Things Ecosystem*. [S.l.]: Elsevier, 2021. p. 1–20. 13

JAIN, S. M. Namespaces. In: _____. *Linux Containers and Virtualization: A Kernel Perspective*. Berkeley, CA: Apress, 2020. p. 31–43. ISBN 978-1-4842-6283-2. Disponível em: <https://doi.org/10.1007/978-1-4842-6283-2_3>. 54, 56

JUN, H.-J.; YANG, H.-S. Performance of the xmpp and the mqtt protocols on iec 61850-based micro grid communication architecture. *Energies*, MDPI, v. 14, n. 16, p. 5024, 2021. 28, 34

KHALIL, K. E. K. et al. Resource discovery techniques in the internet of things: A review. *Internet of Things*, Elsevier, v. 12, p. 100293, 2020. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2542660520301256>. 22, 23, 24

KHOROV, E. et al. A tutorial on ieee 802.11 ax high efficiency wlans. *IEEE Communications Surveys & Tutorials*, IEEE, v. 21, n. 1, p. 197–216, 2018. 19

KIM, K. et al. What's your protocol: Vulnerabilities and security threats related to z-wave protocol. *Pervasive and Mobile Computing*, Elsevier, v. 66, p. 101211, 2020. 41

KIM, T. et al. A study of the z-wave protocol: implementing your own smart home gateway. In: IEEE. *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*. [S.l.], 2018. p. 411–415. 40

LABIOD, H.; AFIFI, H.; SANTIS, C. D. *Wi-Fi, bluetooth, zigbee and wimax*. [S.l.]: Springer, 2007. v. 1. 20

LILLI, M. et al. Formal proof of a vulnerability in z-wave iot protocol. In: *SECRYPT*. [S.l.: s.n.], 2021. p. 198–209. 40, 41

Linux Foundation. *Iproute2*. 2023. Disponível em: <https://wiki.linuxfoundation.org/networking/iproute2>. Acesso em: 30 Maio 2024. 60

LOPEZ-PAVON, C.; SENDRA, S.; VALENZUELA-VALDÉS, J. F. Evaluation of cupcarbon network simulator for wireless sensor networks. *Network Protocols and Algorithms*, Macrothink Institute, v. 10, n. 2, p. 1–27, 2018. 45

MACEDO, R. da S. *Um estudo e avaliação do desempenho de Protocolos de Aplicação para a Internet das Coisas*. Dissertação (Mestrado) — Mestrado. Programa de Pós-Graduação em Ciências Computacionais, Universidade do Estado do Rio de Janeiro, 2019. Disponível em: <http://www.bdtd.uerj.br/handle/1/17641>. 51

MAGRANI, E. *A internet das coisas*. [S.l.]: BOD GmbH DE, 2021. 15

Man7 Org. *namespaces(7) — Linux manual page*. 2024. Disponível em: <https://man7.org/linux/man-pages/man7/namespaces.7.html>. Acesso em: 09 Agosto 2024. 54

MANIVANNAN, T.; RADHAKRISHNAN, P. A comprehensive analysis of simulation tools for internet of things. *Solid State Technology*, v. 63, n. 5, p. 461–471, 2020. 46, 47

MARKSTEINER, S. et al. An overview of wireless iot protocol security in the smart home domain. *2017 Internet of Things Business Models, Users, and Networks*, IEEE, p. 1–8, 2017. 41

MARKUS, A.; KERTESZ, A. A survey and taxonomy of simulation environments modelling fog computing. *Simulation Modelling Practice and Theory*, Elsevier, v. 101, p. 102042, 2020. 44, 45

MCATEER, I. N. et al. Security vulnerabilities and cyber threat analysis of the amqp protocol for the internet of things. 2017. 30

Mosquitto Org. *Eclipse Mosquitto - An open source MQTT broker*. Disponível em: <https://mosquitto.org>. Acesso em: 30 Maio 2024. 59

Mozilla Org. *Uma visão geral do HTTP*. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 30 setembro 2023. 32

MQTT Org. *MQTT Specifications*. 2022. Disponível em: <https://mqtt.org/mqtt-specification/>. Acesso em: 30 setembro 2023. 26

MQTT Org. *MQTT: The Standard for IoT Messaging*. 2022. Disponível em: <https://mqtt.org>. Acesso em: 30 setembro 2023. 13, 23

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: IEEE. *2017 IEEE international systems engineering symposium (ISSE)*. [S.l.], 2017. p. 1–7. 32, 33, 34

NIKOLOV, N. Research of mqtt, coap, http and xmpp iot communication protocols for embedded systems. In: IEEE. *2020 XXIX international scientific conference electronics (ET)*. [S.l.], 2020. p. 1–4. 28, 32

OOKO, S. O. et al. Security issues in ipv6 over low-power wireless personal area networks (6lowpan): a review. In: IEEE. *2020 21st International Arab Conference on Information Technology (ACIT)*. [S.l.], 2020. p. 1–5. 36, 37

PADGETTE, J.; SCARFONE, K.; CHEN, L. Guide to bluetooth security. *NIST special publication*, v. 800, n. 121, 2017. 13, 20

PAPADOPOULOS, G. Z. et al. Adding value to wsn simulation using the iot-lab experimental platform. In: IEEE. *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. [S.l.], 2013. p. 485–490. 43

PAPADOPOULOS, G. Z. et al. Thorough iot testbed characterization: From proof-of-concept to repeatable experimentations. *Computer Networks*, Elsevier, v. 119, p. 86–101, 2017. 43, 44

PARK, P.; FISCHIONE, C.; JOHANSSON, K. H. Modeling and stability analysis of hybrid multiple access in the ieee 802.15. 4 protocol. *ACM Transactions on Sensor Networks (TOSN)*, ACM New York, NY, USA, v. 9, n. 2, p. 1–55, 2013. 38

PATEL, N.; MEHTRE, B.; WANKAR, R. Simulators, emulators, and test-beds for internet of things: A comparison. In: IEEE. *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. [S.l.], 2019. p. 139–145. 43, 44, 45, 46, 47, 48, 49, 50

PATHAKA, A. D.; TEMBHURNE, J. V. Internet of things: a survey on iot protocols. In: *Proceedings of 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT)*. [S.l.: s.n.], 2018. p. 26–27. 29

PFLANZNER, T. et al. Mobiotsim: Towards a mobile iot device simulator. In: IEEE. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. [S.l.], 2016. p. 21–27. 46

Pypi Org. *netifaces 0.11.0*. 2021. Disponível em: <https://pypi.org/project/netifaces/>. Acesso em: 09 Agosto 2024. 59

QUINCOZES, V. E.; QUINCOZES, S. E.; KAZIENKO, J. F. Desvendando a camada de aplicaçao na internet das coisas: Teoria, prática e tendências. *Sociedade Brasileira de Computação*, 2021. 34

RAMIREZ, J.; PEDRAZA, C. Performance analysis of communication protocols for internet of things platforms. In: IEEE. *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*. [S.l.], 2017. p. 1–7. 51, 87

RAMYA, C. M.; SHANMUGARAJ, M.; PRABAKARAN, R. Study on zigbee technology. In: IEEE. *2011 3rd international conference on electronics computer technology*. [S.l.], 2011. v. 6, p. 297–301. 13, 20

RASHID, B. N.; SHARIF, K. H.; SHREEF, T. H. A survey of simulation tools for modelling internet of thing. *Passer Journal of Basic and Applied Sciences*, University of Garmian, v. 4, n. 1, p. 37–44, 2022. 45, 47

ROLDÁN-GÓMEZ, J. et al. Security analysis of the mqtt-sn protocol for the internet of things. *Applied Sciences*, MDPI, v. 12, n. 21, p. 10991, 2022. 26

ROSEN, R. *Linux kernel networking: Implementation and theory*. [S.l.]: Apress, 2014. 56

SADIO, O.; NGOM, I.; LISHOU, C. Lightweight security scheme for mqtt/mqtt-sn protocol. In: IEEE. *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. [S.l.], 2019. p. 119–123. 26

SALMAN, T.; JAIN, R. A survey of protocols and standards for internet of things. *arXiv preprint arXiv:1903.11549*, 2019. 39, 40

SANTOS, B. P. et al. Internet das coisas: da teoria à prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, v. 31, p. 16, 2016. 13

Scapy Community. *Scapy*. 2024. Disponível em: <https://scapy.net/>. Acesso em: 09 Agosto 2024. 59

SEMJONOV, A. *Security analysis of user namespaces and rootless containers*. Tese (Doutorado), 2020. 56

SERRANO, M. et al. Fiestaiot project: Federated interoperable semantic iot/cloud testbeds and applications. In: *Companion Proceedings of the The Web Conference 2018*. [S.l.: s.n.], 2018. p. 425–426. 48

SILVA, D. et al. A performance analysis of internet of things networking protocols: Evaluating mqtt, coap, opc ua. *Applied Sciences*, MDPI, v. 11, n. 11, p. 4879, 2021. 51, 87, 88, 90

SINGH, A.; NANDANWAR, H.; CHAUHAN, A. Simulation tools and testbeds for internet of things (iot):"comparative insight". In: IEEE. *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*. [S.l.], 2022. p. 1–7. 46, 47, 48, 49

SOBIN, C. A survey on architecture, protocols and challenges in iot. *Wireless Personal Communications*, Springer, v. 112, n. 3, p. 1383–1429, 2020. 15, 16, 17

SOBRAL, J. V. et al. Routing protocols for low power and lossy networks in internet of things applications. *Sensors*, MDPI, v. 19, n. 9, p. 2144, 2019. 35, 36

STANIK, A.; KAO, O. A proposal for rest with xmpp as base protocol for intercloud communication. In: IEEE. *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*. [S.l.], 2016. p. 1–6. 28, 29

STUSEK, M. et al. Iot protocols for low-power massive iot: a communication perspective. In: IEEE. *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. [S.l.], 2019. p. 1–7. 21, 22

SYED, M. I. et al. The challenges of trace-driven wi-fi emulation. *arXiv preprint arXiv:2002.03905*, 2020. 48

Tcpdump Org. *TCPDUMP LiBPCAP*. Disponível em: <https://www.tcpdump.org>. Acesso em: 30 Maio 2024. 59

THUNGON, L. C. et al. A lightweight authentication and key exchange mechanism for ipv6 over low-power wireless personal area networks-based internet of things. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 32, n. 5, p. e4033, 2021. 36, 37

TOLCHA, Y.; KIM, D. Distributed event-based resource-oriented epcglobal middleware using gpc. In: IEEE. *2016 IEEE International Conference on RFID Technology and Applications (RFID-TA)*. [S.l.], 2016. p. 29–34. 39

TSOUTSA, P.; FITSILIS, P.; RAGOS, O. Service simulation in industry 4.0: A comparison of simulators. *Industry 4.0*, Scientific Technical Union of Mechanical Engineering"Industry 4.0", v. 3, n. 2, p. 101–104, 2018. 44

TYAGI, P.; DEMBLA, D. Performance analysis and quality-of-service monitoring of protected and unprotected tcp networks using nctuns simulator. In: IEEE. *2015 Fifth International Conference on Communication Systems and Network Technologies*. [S.l.], 2015. p. 273–277. 50

VAIGANDLA, K. K.; KARNE, R. K.; RAO, A. S. A study on iot technologies, standards and protocols. *IBMRD's Journal of Management & Research*, v. 10, n. 2, p. 7–14, 2021. 40

Wireshark Org. *The world's most popular network protocol analyzer*. Disponível em: <https://www.wireshark.org>. Acesso em: 30 Maio 2024. 59

WUKKADADA, B. et al. Comparison with http and mqtt in internet of things (iot). In: IEEE. *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. [S.l.], 2018. p. 249–253. 32

XIE, D.; LI, J.; GAO, H. Comparison and analysis of simulation methods for tsn performance. In: IOP PUBLISHING. *IOP Conference Series: Materials Science and Engineering*. [S.l.], 2020. v. 768, n. 5, p. 052061. 45

XMPP Org. *The universal messaging standard*. 2023. Disponível em: <https://xmppt.org>. Acesso em: 30 setembro 2023. 28

YOKOTANI, T.; SASAKI, Y. Comparison with http and mqtt on required network resources for iot. In: IEEE. *2016 international conference on control, electronics, renewable energy and communications (ICCEREC)*. [S.l.], 2016. p. 1–6. 33

YUGHA, R.; CHITHRA, S. A survey on technologies and security protocols: Reference for future generation iot. *Journal of Network and Computer Applications*, v. 169, p. 102763, 2020. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S108480452030237X>. 29, 31