UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

MIGUEL ANGELO ROMANICHEN SUCHODOLAK

DESENVOLVIMENTO DE UM GATEWAY DE COMUNICAÇÃO MQTT E OPC-UA PARA DISPOSITIVOS DE COMPUTAÇÃO DE BORDA

Joinville
2024

MIGUEL ANGELO ROMANICHEN SUCHODOLAK

DESENVOLVIMENTO DE UM GATEWAY DE COMUNICAÇÃO MQTT E OPC-UA PARA DISPOSITIVOS DE COMPUTAÇÃO DE BORDA

Trabalho apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica, no Curso de Engenharia Mecatrônica, do Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina.

Orientador: Dr. Gian Ricardo Berkenbrock

Joinville
2024

Dedico este trabalho a meus pais Lucia e Waldemar

AGRADECIMENTOS

Agradeço de coração aos meus pais, Lucia e Waldemar, por terem acreditado em mim e por se esforçarem tanto para que eu pudesse ter a educação que recebi.

Agradeço ao laboratório LSE pelos equipamentos disponibilizados para que esse trabalho fosse executado. Agradeço também a todos os membros do laboratório, que sempre que puderam oferecer sua ajuda.

Ao meu orientador, Dr. Gian Ricardo Berkencrock, por sua orientação valiosa não apenas na elaboração deste trabalho, mas também em decisões importantes para minha carreira acadêmica e profissional.

Por fim, quero expressar minha gratidão a todos os meus amigos que fizeram parte da minha jornada até aqui. Sem vocês tudo isso não seria a mesma coisa.

Efficiency is doing things right, effectiveness is doing the right things (Drucker, 2001).

RESUMO

Em um contexto industrial cada vez mais tecnológico e interconectado, a integração de novas tecnologias muitas vezes é marcada por uma implementação desordenada, caracterizada pela combinação de tecnologias antigas com tecnologias de última geração. Com o aumento exponencial na geração de dados devido à proliferação de dispositivos conectados à internet e principalmente dispositivos inteligentes usados na indústria, torna-se crucial ordenar e priorizar a transmissão dessas informações, considerando não apenas as limitações de largura de banda, mas também a latência e a segurança dos dados em transmitidos. Nesse sentido, dispositivos de computação de borda desempenham um papel fundamental na gestão e segurança dos dados, permitindo o processamento próximo à fonte de geração. No entanto, a integração desses dispositivos muitas vezes enfrenta desafios significativos, especialmente em ambientes sensíveis à interferência externa, como fábricas e instalações industriais, onde a estabilidade e a confiabilidade são essenciais para o funcionamento seguro e eficiente dos processos, pois muitas vezes a variedade de protocolos de comunicação usados, assim como uma grande diversidade de dispositivos IIoT. Diante desse cenário, utilizou-se neste trabalho s protocolos MQTT e OPC-UA para o desenvolvimento de um sistema de computação de borda responsável pela comunicação e validação de dados entre dispositivos IIoT e um gateway de saída. Além da comunicação entre dispositivos distribuídos, a proposta inclui a conexão direta com um banco de dados e o software de visualização Thingsboard, viabilizando a armazenagem e o processamento dos dados coletados de maneira segura e escalável.

Palavras-chave: IoT; monitoramento de dados; MQTT; OPC-UA.

ABSTRACT

In an increasingly technological and interconnected industrial context, the integration of new technologies is often marked by a disordered implementation, characterized by the combination of outdated technologies with cutting-edge ones. With the exponential increase in data generation due to the proliferation of internet-connected devices, and especially smart devices used in industry, it becomes crucial to organize and prioritize the transmission of this information, considering not only bandwidth limitations but also latency and data security during transmission. In this regard, edge computing devices play a fundamental role in data management and security, enabling processing close to the source of generation. However, the integration of these devices often faces significant challenges, particularly in environments sensitive to external interference, such as factories and industrial facilities, where stability and reliability are essential for the safe and efficient operation of processes. This is often compounded by the vast variety of communication protocols used and the diversity of IIoT devices. Given this scenario, this work proposes the use of MQTT and OPC-UA protocols for the development of an edge computing system responsible for communication between IIoT devices and an output gateway. In addition to communication between distributed devices, the proposal includes direct connection to a database and the ThingsBoard visualization software, enabling secure and scalable storage and processing of the collected data.

Keywords: IoT; data monitoring; MQTT; UPC-UA.

LISTA DE FIGURAS

Figura 1 – Nove pilares da indústria 4.0 17
Figura 2 – Esquema de Funcionamento do protocolo OPC 20
Figura 3 – Sentido de comunicação OPC-UA 22
Figura 4 – Camadas das especificações OPC-UA 22
Figura 5 – Camadas do cliente OPC-UA 23
Figura 6 – Estrutura de um nó 24
Figura 7 – Camadas do servidor OPC-UA 25
Figura 8 – Esquema OPC-UA PubSub 26
Figura 9 – Esquema de Funcionamento do protocolo MQTT 27
Figura 10 – QoS 0 no protocolo MQTT 31
Figura 11 – QoS 1 no protocolo MQTT 31
Figura 12 – QoS 2 no protocolo MQTT 32
Figura 13 – Computação na borda 34
Figura 14 – Conceito de operação do sistema 37
Figura 15 – Diagrama do fluxo de dados 40
Figura 16 – Diagrama consumidor 42
Figura 17 – Mapeamento dos processos 43
Figura 18 – Esquema FIFO entrada 45
Figura 19 – Esquema FIFO consumidores 46
Figura 20 – Esquema estrutura JSON 47
Figura 21 – Esquema de isolação VM - Hospedeiro 48
Figura 22 – Resultado teste de latência 50
Figura 23 – Resultado consumo de RAM 52
Figura 24 – Tamanho de fila conforme mudança da frequência 54
Figura 25 – Resultado do teste com adição de latência 55
Figura 26 – Resultado teste de latência 57
Figura 27 – Resultado teste de latência 58
Figura 28 – Consumo de RAM durante teste de latência 59
Figura 29 – Tamanho de fila conforme alteração na frequência de envio 61
Figura 30 – Consumo de RAM durante teste de vazão. 62
Figura 31 – Resultado do teste com adição de latência 63
Figura 32 – Consumo de RAM durante teste de perdas 65

LISTA DE QUADROS

Quadro 1 – Pacotes de controle MQTT. 28
Quadro 2 – Cabeçalho fixo MQTT. 29
Quadro 3 – Relação de QoS com Pub/Sub. 30

LISTA DE TABELAS

Tabela 1 – Dados de latências em ms, correlacionado ao tamanho das mensagens 51
Tabela 2 – Dados de uso de CPU em relação ao tamanho das mensagens . . 52
Tabela 3 – Vazão total do software em kilobytes por segundo 53
Tabela 4 – Dados do teste com injeção de perda de pacotes 56
Tabela 5 – Dados de latências em ms, correlacionado ao tamanho das mensagens 57
Tabela 6 – Dados de uso de CPU em relação ao tamanho das mensagens . . 59
Tabela 7 – Vazão total do software em kilobytes por segundo 60
Tabela 8 – Dados do teste com injeção de perda de pacotes 63
Tabela 9 – Dados do teste conexão instável 66

LISTA DE SÍMBOLOS

$IIoT$	Industrial Internet of Things
$MQTT$	Message Queue Telemetry Transport
OPC	Open Plataform Communications
$OPCUA$	Open Plataform Communications - Unified Architecture
$DCOM$	Distributed Component Object Model
TCP	Transmission Control Protocol
$HTTP$	Hypertext Transfer Protocol
XML	Extensible Markup Language
DA	Data Access
AC	Alarms and Conditions
HA	Historical Access
API	Application Programming Interface
$PubSub$	Publish and Subscribe
MOM	Message Oriented Middleware
UDP	User Datagram Protocol
$AMQP$	Advanced Message Queuing Protocol
QoS	Quality of Service
ACK	Acknowledgement
TLS	Transmport Layer Security
SSL	Secure Sockets Layer
IOT	Internet of Things
RMS	Reconfigurable Manufacturing System
CLP	Controlador Lógico Programável
CRC	Cyclic Redundancy Check

RF	Requisito Funcional
RNF	Requisito não Funcional
IDE	Ambiente de Desenvolvimento Integrado
$JSON$	JavaScript Object Notation
$FIFO$	First In First Out
GB	Gigabyte
KB	Kilobyte

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 16
1.1.1 Objetivo Geral 16
1.1.2 Objetivos Específicos 16

2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 INDÚSTRIA 4.0 17
2.2 Protocolo Open Platform Communications 19
2.2.1 Protocolo OPC - Unified Architecture 21
2.2.1.1 Funcionamento OPC-UA 21
2.2.1.2 OPC-UA PubSub 25
2.3 Protocolo Message Queue Telemetry Transport 27
2.3.1 Estruturação de um pacote MQTT 28
2.3.2 Níveis de QoS no MQTT 29
2.4 COMPUTAÇÃO DE BORDA 33
2.5 TRABALHOS RELACIONADOS 35

3 MATERIAIS E MÉTODOS 36
3.1 CONCEITO DE OPERAÇÃO 36
3.2 REQUISITOS DE PROJETO 37
3.2.1 Requisitos funcionais 37
3.2.2 Requisitos não funcionais 39
3.3 Gerenciamento de fluxo 40
3.4 Modelagem 41
3.4.1 Mapeamento dos processos 42
3.5 Implementação 44
3.5.1 Comunicação entre processos 45
3.5.2 Estruturação da mensagem 46

4 RESULTADOS E DISCUSSÕES 48
4.1 Testes 48
4.2 Cenário com consumidor único 49
4.2.1 Estudo sobre latência 49
4.2.2 Estudo de vazão total 52
4.2.3 Alteração da Latência 54
4.2.4 Alteração da perda de pacotes 55
4.3 Cenário com consumidores múltiplos 56

4.3.1 Estudo sobre latência 57
4.3.2 Estudo sobre vazão total 59
4.3.3 Alteração da Latência 62
4.3.4 Alteração da perda de pacotes 63
4.4 Cenário simulação 64
4.5 Considerações Parciais 66
4.6 Limitações 67

5 CONCLUSÕES 68

REFERÊNCIAS 70

1 INTRODUÇÃO

Na era da Indústria 4.0 se testemunha uma metamorfose industrial, onde a fusão da cibernética, Industrial Internet of Things (IIoT) e computação em nuvem, redefinem os fundamentos da produção. Esse fenômeno transcende a mera automação, moldando uma nova era de eficiência e inovação disruptiva (Schwab, 2016).

A integração de sistemas ciber-físicos e análise de dados em tempo real não apenas otimiza a eficiência operacional, mas também inaugura possibilidades de personalização em massa. Nesse contexto, observa-se uma mudança drástica na manufatura, onde a conectividade e a inteligência artificial convergem para redefinir a essência da produção (Porter; Heppelmann, 2014).

Além disso, destaca-se que a Indústria 4.0 não se limita apenas ao chão de fábrica, mas se estende por toda a cadeia de valor, desde a concepção do produto até a entrega ao consumidor. Essa abrangência impacta não apenas a eficiência, mas também a flexibilidade e a capacidade de inovação das organizações, resultando em uma reconfiguração completa da atual fase da indústria (Brynjolfsson; McAfee, 2014).

Em meio a esta revolução, os dispositivos IIoT emergem como peças essenciais para o dinamismo da produção, pois, ao coletarem e transmitirem dados em tempo real, esses dispositivos oferecem informações valiosas para aprimorar continuamente os processos industriais. A capacidade de analisar e interpretar dados provenientes de máquinas interconectadas impulsiona o aumento da produtividade e também se torna um diferencial estratégico para a manutenção proativa, antecipando potenciais falhas e otimizando a confiabilidade do sistema (Brynjolfsson; McAfee, 2014).

Com base na necessidade do monitoramento desses dados, propõe-se, neste trabalho, a adoção de um sistema de computação de borda eficiente, utilizando os protocolos Message Queuing Telemetry Transport (MQTT) e Open Platform Communications Unified Architecture (OPC-UA). Esse sistema visa garantir o recebimento de dados provenientes de dispositivos IIoT, promovendo uma visão do ambiente industrial por meio da implementação de uma comunicação de ponte com o software de visualização Thingsboard, o qual possui um banco de dados integrado.

A metodologia adotada para a consecução dos objetivos propostos compreende uma análise preliminar visando à definição da estrutura e formatação dos dados a serem transmitidos. A implementação também propõe uma solução apta a suportar, de forma simultânea, os protocolos MQTT e OPC-UA, assegurando a eficácia da comunicação entre os dispositivos IIoT e o sistema de computação de borda. A viabilidade de análises aprofundadas sobre o tempo de resposta e o desenvolvimento de modelos estatísticos sobre o funcionamento do sistema pauta-se na condução de testes laboratoriais a fim de validar experimentalmente os resultados obtidos e avaliar o comportamento

operacional do sistema.

1.1 OBJETIVOS

Para resolver a problemática dos monitoramentos de dados provenientes de dispositivos IIoT e formalizar sua validade, propõe-se neste trabalho os seguintes objetivos.

1.1.1 Objetivo Geral

Elaborar uma aplicação que funcione como uma interface entre diversos dispositivos IIoT e um banco de dados, visando garantir a entrega dos dados a serem transferidos.

1.1.2 Objetivos Específicos

- Definir os dados, que deverão ser transmitidos;
- Projetar uma solução para lidar com protocolos a serem definidos;
- Implementar uma solução capaz de suportar o processamento dos dados transmitidos
- Avaliar os resultados obtidos em testes;

2 FUNDAMENTAÇÃO TEÓRICA

Visando compreender os desafios da computação de borda em ambientes industriais, é necessário a conceitualização de um ambiente industrial na indústria 4.0, assim como os dispositivos presentes nesse meio. Apresenta-se neste capítulo, os conceitos que envolvem a transferência de armazenamento de dados, entre os quais a caracterização do protocolo OPC-UA e MQTT, assim como as características da indústria 4.0 e de dispositivos de computação de borda.

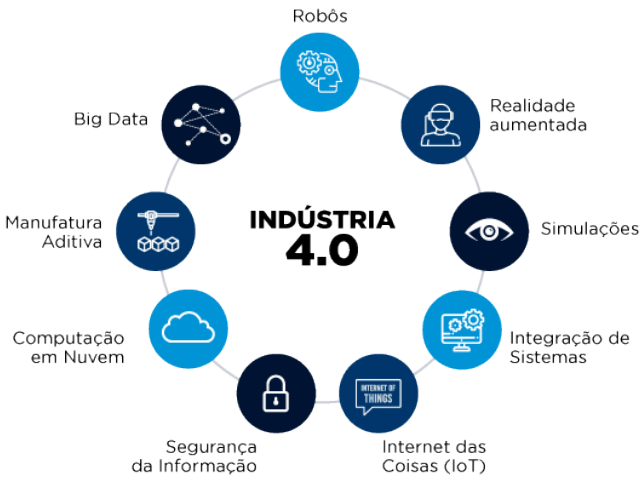
2.1 INDÚSTRIA 4.0

A evolução da indústria tem sido marcada por quatro revoluções industriais com características únicas. A primeira revolução industrial ocorreu durante o século XVIII, e foi principalmente marcada pela utilização de energia a vapor e produção no modelo de manufatura (Smith, 2020).

Já a segunda pode ser destacada pela produção em massa e uso da eletricidade. A terceira iniciou no século XIX, sendo demarcada pelo uso da eletrônica e microeletrônica. Por último, vem a quarta e atual revolução, caracterizada especialmente pelos conceitos da indústria 4.0 (Coelho, 2016).

Segundo Coelho (2016), a indústria 4.0 tem nove pilares de funcionamento, apresentados na Figura 1. Embora essas tecnologias já sejam empregadas de maneira individual no setor industrial, sua integração propicia a assimilação de células distintas, otimizando o fluxo de produção e aumentando a eficiência produtiva.

Figura 1 – Nove pilares da indústria 4.0



Fonte: Chesini (2021, p. 1).

Os sistemas autônomos, especialmente veículos e robôs autônomos, desempenham um papel central na Indústria 4.0, transformando a dinâmica dos ambientes industriais. Projetados para realizar tarefas complexas sem intervenção humana direta, esses sistemas promovem eficiência operacional e segurança (Smith, 2020).

Sua capacidade de navegação autônoma, análise de dados em tempo real e adaptação a ambientes dinâmicos representam uma vantagem na otimização de processos de produção e logística. A integração desses sistemas não só impulsiona avanços tecnológicos, mas também confere um diferencial na busca por operações mais rápidas e adaptáveis (Smith, 2020).

O Big Data é um dos pilares fundamentais na arquitetura da Indústria 4.0, ele se baseia na responsabilidade da gestão e análise de grandes volumes de dados gerados por sistemas interconectados. Uma vez que, a capacidade de lidar com conjuntos massivos de informações em tempo real oferece uma vantagem significativa para otimizar processos industriais e embasar decisões correlacionadas a direção que a produção irá tomar (Wang, 2018).

A realidade aumentada, por sua vez, facilita a integração de dados e propiciar novas modalidades de acesso à informação, como exemplos de atividades que se beneficiam da implementação da realidade aumentada pode-se citar manuais de reparo, seleção de equipamentos e treinamentos virtuais de colaboradores. Essa tecnologia proporciona uma capacidade de enriquecer a percepção do usuário com sobreposições de informações digitais, aprimorando, assim, a execução de diversas tarefas no contexto industrial (Guyon, 2019).

A manufatura aditiva, ou impressão 3D, é uma tecnologia disruptiva da Indústria 4.0 que cria objetos tridimensionais camada por camada a partir de modelos digitais (Calzado *et al*., 2019). Essa abordagem proporciona uma capacidade além da fabricação tradicional por permitir a produção de peças complexas e personalizadas, que por meio de aplicações variadas, desde prototipagem rápida até a produção em escala para setores como aeroespacial e saúde. Isso introduz velocidade na cadeia produtiva, proporciona redução de desperdícios e maior liberdade de design para os produtos (Smith, 2020).

As simulações dentro Indústria 4.0, possibilitam a análise virtual de processos industriais complexos para otimização e tomada de decisões eficientes antes mesmo de se ter determinado produto em mãos (Ferreira *et al*., 2021). Essa abordagem, comparada aos métodos convencionais, permite a modelagem precisa e a avaliação de cenários variados, desde o projeto de layout de fábrica até a previsão de desempenho operacional, sem ter o acesso direto a linha de produção (Guyon, 2019).

A computação em nuvem, por sua vez, proporciona recursos computacionais escaláveis e acessíveis conforme a demanda. Esta tecnologia permite armazenar

grandes volumes de dados, processamento distribuído e acesso remoto a serviços, uma vez que muitas vezes a rede cabeada não supri a necessidade de muitos ambientes industriais (Smith, 2020).

A integração de sistemas na Indústria 4.0 é considerado um dos maiores desafios dessa revolução, contudo, a conexão fluida entre diferentes componentes ciber-físicos propõe a harmonização de processos e dados, promovendo a interoperabilidade e integração produtiva (Lee, 2015). Ao superar as barreiras de comunicação entre sistemas heterogêneos, a integração de sistemas possibilita uma visão unificada das operações industriais (Finance, 2015).

A segurança da informação na indústria 4.0, dada a interconexão intensiva de dispositivos nesse ambiente da indústria 4.0, demanda robustas medidas de proteção para salvaguardar dados sensíveis e assegurar a continuidade dos sistemas. Estratégias como criptografia avançada, autenticação multifatorial e monitoramento constante são essenciais para mitigar potenciais ameaças cibernéticas. A integração de protocolos de segurança em todas as camadas da arquitetura ciber-física garante a confidencialidade, integridade e disponibilidade dos dados (Kang, 2016).

A Internet das Coisas Industriais na Indústria 4.0 atua como facilitadora na integração inteligente entre dispositivos e sistemas industriais. Esta rede de sensores, máquinas e dispositivos opera em tempo real, proporcionando uma visão instantânea das operações industriais. A IIoT desempenha um papel crucial ao permitir a coleta contínua de dados, impulsionando o monitoramento remoto, manutenção preditiva e otimização de processos, pode também melhorar o tempo de resposta para a atuação dos sistemas envolvidos (Xu; He; Li, 2014).

2.2 PROTOCOLO OPEN PLATFORM COMMUNICATIONS

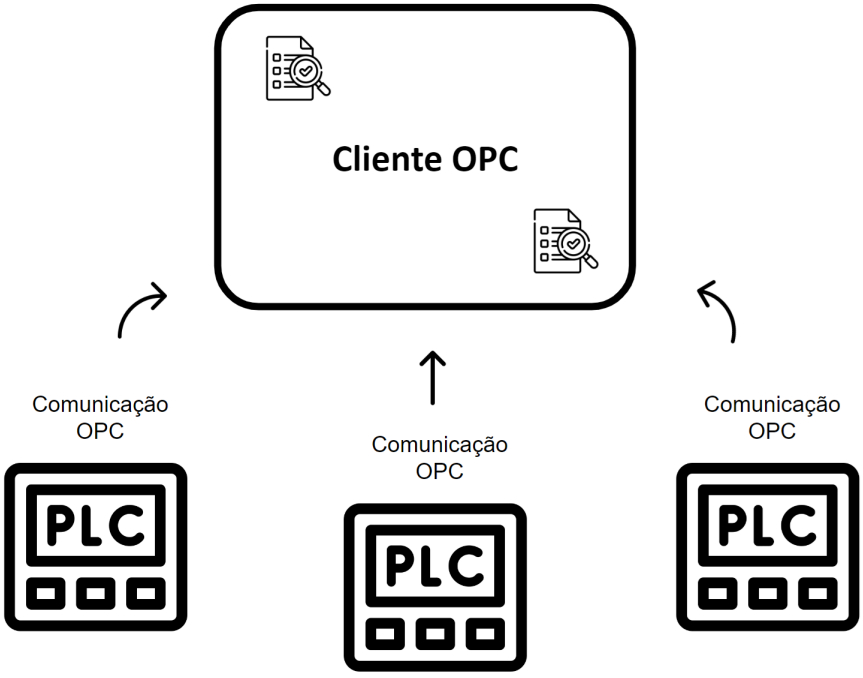
Em meados de 1995, um marco na evolução da interoperabilidade em ambientes industriais foi estabelecido através da colaboração entre diversas empresas, essa iniciativa culminou na criação da Open Platform Communications (OPC) Foundation, uma entidade que desenvolveu o padrão OPC. No contexto industrial o qual o OPC foi criado, evidenciava-se a necessidade de interoperabilidade na indústria, assim como a convergência de esforços para estabelecer um padrão que suporta a comunicação entre o sistema Windows e dispositivos industriais (Rocha, 2013).

Com a participação ativa de membros da Microsoft, ao proporcionar suporte técnico à solução proposta, consolidou-se a base para a troca segura e confiável de dados no domínio da automação industrial. Essa colaboração foi responsável pela criação de um protocolo que integra a comunicação de praticamente qualquer controlador lógico, independentemente do fabricante, ao sistema Windows (Rocha, 2013).

O Protocolo OPC representa no cenário da automação industrial, um padrão para a troca de dados segura e eficiente. Sua contribuição para a integração de sistemas, coordenação de operações e fluxo contínuo de informações consolida sua posição como uma tecnologia essencial na automação industrial em diversas indústrias (Rocha, 2013).

Esse protocolo, utiliza a tecnologia Distributed Component Object Model (DCOM) da Microsoft como base para a comunicação entre sistemas. Sua funcionalidade está centrada na arquitetura cliente-sevidor, demonstrado na Figura 2. O servidor OPC, que pode ser integrado a dispositivos industriais, sistemas de controle ou outros sistemas de automação, disponibiliza dados e funcionalidades específicos para outros sistemas na rede. Por outro lado, os clientes OPC consomem esses dados e funcionalidades para realizar operações como leitura, escrita e monitoramento (Hellsten; Reponen, 2016).

Figura 2 – Esquema de Funcionamento do protocolo OPC



Fonte: Elaborado pelo autor (2023)

Essencialmente, o OPC Clássico atua como uma interface de comunicação, traduzindo entre os diversos protocolos utilizados por dispositivos e sistemas na rede. Isso permite uma comunicação simples mesmo em ambientes onde diferentes protocolos são empregados (Husemann; Roß; Vossebein, 1999).

2.2.1 Protocolo OPC - Unified Architecture

Apesar de suas contribuições, o OPC Clássico apresenta limitações, especialmente a dependência do DCOM. Essa dependência pode resultar em desafios relacionados à segurança e integração de sistemas, especialmente em ambientes onde a comunicação dos dispositivos ocorre com servidores diferentes do sistema Windows. Essas limitações impulsionaram o desenvolvimento do Unified Architecture (UA), uma evolução projetada para superar as lacunas do OPC Clássico e atender às crescentes demandas da Indústria 4.0 (Rocha, 2013).

Ao contrário do OPC clássico, que focava na comunicação entre sistemas Windows, o OPC-UA amplia seu escopo para oferecer conectividade entre diferentes sistemas operacionais e arquiteturas de hardware. Sua arquitetura orientada a serviços permite a troca segura e confiável de dados, suportando não apenas a comunicação em tempo real, mas também a integração de serviços como alarmes, eventos e gerenciamento de segurança (Jadala *et al*., 2021).

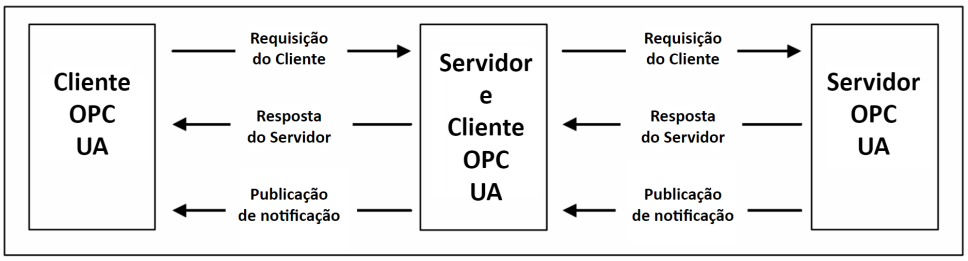
Além disso, o OPC-UA incorpora mecanismos avançados de segurança, como criptografia e autenticação, abordando preocupações críticas em ambientes industriais. A modelagem de informações padronizada e a capacidade de representar estruturas complexas de dados contribuem para a flexibilidade e adaptabilidade do protocolo em diferentes contextos industriais (Jadala *et al*., 2021).

2.2.1.1 Funcionamento OPC-UA

Embora apresente variações, o OPC-UA mantém, em sua essência, a mesma base de funcionamento de seu antecessor, estando fundamentado na estrutura cliente-servidor. Nesse modelo de comunicação, os servidores assumem a responsabilidade pela interação com os dispositivos e pela disponibilização dos dados adquiridos. Por sua vez, os clientes têm a tarefa de comunicar-se com os servidores e acessar os dados por eles publicados (Mahnke; Leitner; Dam, 2010).

Em uma infraestrutura de comunicação baseada no protocolo OPC-UA, cada sistema tem o poder de hospedar múltiplos clientes e servidores. Nesse contexto, cada cliente pode estabelecer conexões simultâneas com um ou mais servidores, e contrário também se faz presente. Conforme ilustrado na Figura 3, um cliente pode solicitar informações a um servidor, que, por sua vez, encaminha a resposta correspondente. Além disso, é possível que um mesmo dispositivo exerça tanto o papel de cliente quanto o de servidor dentro da mesma implementação. Portanto, ele pode tanto requisitar informações de outros sistemas quanto disponibilizar dados para serem acessados por outros clientes no ambiente OPC-UA (Mahnke; Leitner; Dam, 2010).

Figura 3 – Sentido de comunicação OPC-UA

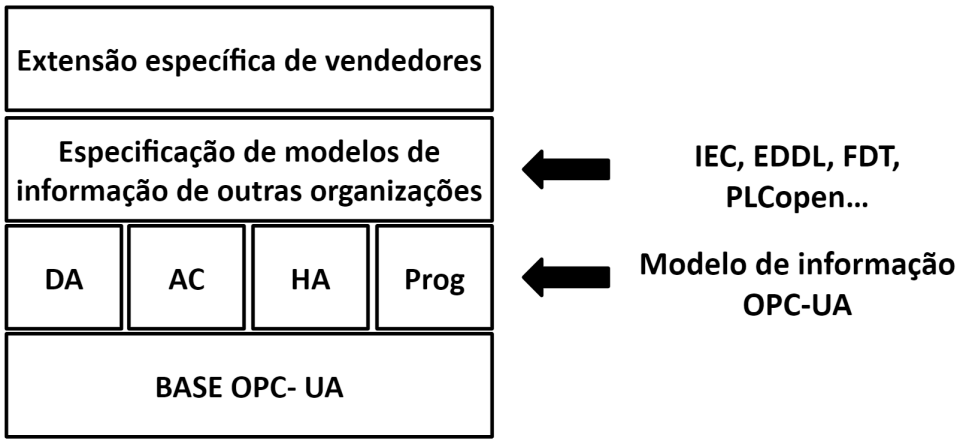


Fonte: Adaptado de (OPC-Foundation, 2021)

A comunicação entre clientes e servidores no contexto do protocolo OPC-UA pode ser analisada com base em dois conceitos fundamentais: a modelagem dos dados e o mecanismo de transporte. O mecanismo de transporte se baseia no uso do protocolo de transmissão de controle de protocolo (TCP) combinado com o mapeamento de padrões como Web Services, XML, protocolo de transferência Hypertext (HTTP) e MQTT.

Por outro lado, a modelagem dos dados, como representado na Figura 4, é estruturada em camadas. Isso permite que o cliente OPC-UA acesse os dados necessários pelos menores pacotes possíveis, sem necessidade de compreender integralmente o modelo do sistema (Mahnke; Leitner; Dam, 2010).

Figura 4 – Camadas das especificações OPC-UA



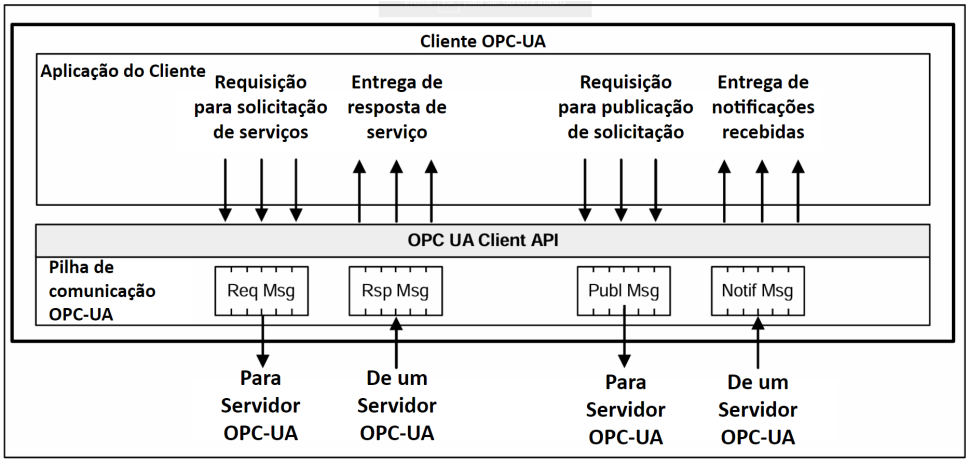
Fonte: Adaptado de (Mahnke; Leitner; Dam, 2010).

Essa modelagem, represa as camadas de abstração em relação a especificações do fabricante do dispositivo, informações de outras organizações, e em relação aos modelos de acesso às informações da base do OPC-UA. Data Access (DA) define as especificações para o acesso aos dados, Alarms and Conditions (AC) representa as especificações dos alarmes disponibilizados na base do OPC UA. Caso o dispositivo tenha configurado o acesso do histórico de dados, o Historical Access

(HA) é a especificação aplicada e Prog define a estrutura para a manipulação de outros programas (Mahnke; Leitner; Dam, 2010).

Na implementação do cliente OPC UA, também é adotada uma estruturação em camadas, dividindo-se em dois grupos distintos: a aplicação cliente e o cliente responsável pela gestão da pilha de comunicação com o servidor OPC UA. Esta organização é ilustrada na Figura 5. No contexto desse modelo de cliente, a comunicação entre as camadas é abstrata para o utilizador da aplicação cliente (OPC-Foundation, 2021).

Figura 5 – Camadas do cliente OPC-UA



Fonte: Adaptado de (OPC-Foundation, 2021).

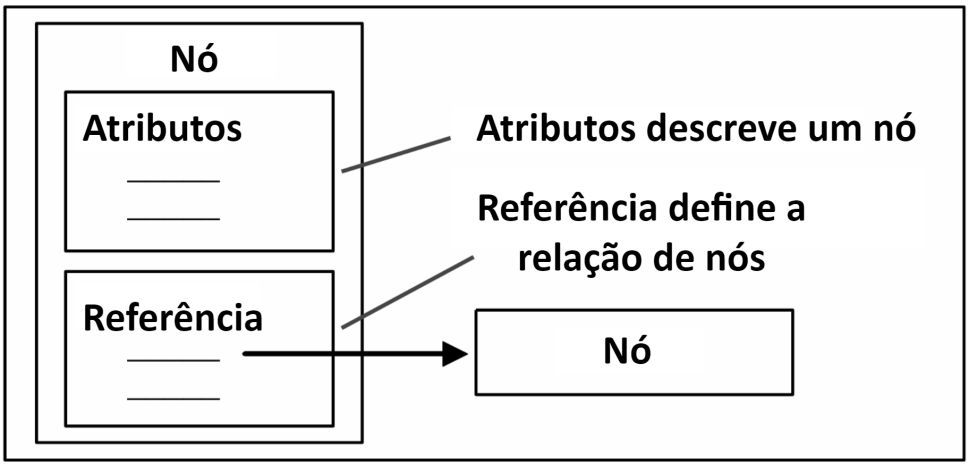
Na estrutura interna da interface, a pilha de comunicação do OPC UA converte as chamadas da interface de aplicação da programação (API) do cliente em mensagens formatadas conforme o padrão OPC UA e as encaminha para o servidor. Nesse contexto, a pilha de comunicação também assume a responsabilidade de gerenciar o recebimento de respostas e a chegada de mensagens de notificação originadas do servidor OPC UA, encarregando-se de entregá-las à camada de aplicação do cliente (OPC-Foundation, 2021).

Para iniciar a comunicação com o servidor e efetuar a leitura ou escrita de dados, o cliente OPC UA necessita realizar primeiramente a descoberta do espaço de endereçamento dos nós. Este espaço é onde residem os nós, os quais se encontram correlacionados aos valores das variáveis específicas dos dispositivos conectados ao servidor. Portanto, os nós, figuram como entidades discretas de dados presentes no servidor OPC UA (OPC-Foundation, 2021).

Conforme ilustrado na Figura 6, a estruturação do espaço de nós no contexto do OPC UA pode ser compreendida como um agrupamento de entidades nodais. Cada nó, por sua vez, é caracterizado por seus atributos e estabelece interconexões com outros nós por meio de referências. Os atributos dos nós, representam elementos de

dados que podem ser acessados pelos clientes do servidor OPC UA, utilizando serviços como leitura e escrita. Através desses serviços, os clientes podem obter informações detalhadas sobre uma determinada variável, incluindo seu tipo, nome e valor associado (OPC-Foundation, 2021).

Figura 6 – Estrutura de um nó



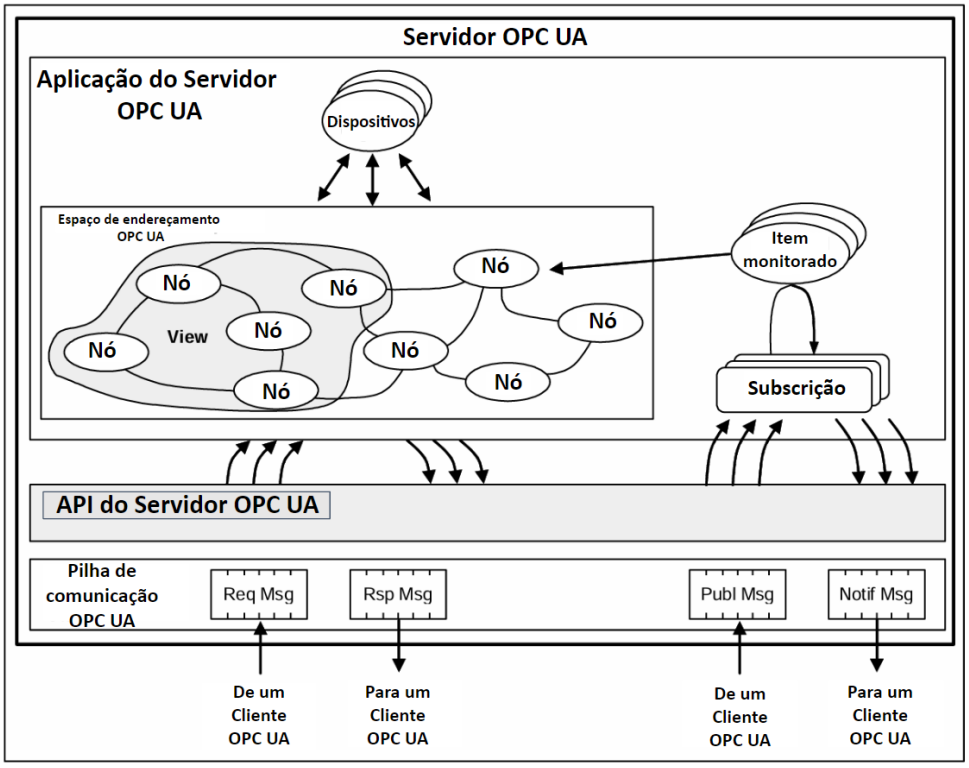
Fonte: Adaptado de (OPC-Foundation, 2021).

O conceito de espaçamento dos nós, constitui um dos elementos na arquitetura do servidor OPC UA. É nesse espaçamento que se estabelece o canal de comunicação com os dispositivos conectados ao servidor, que podem variar desde dispositivos físicos, como controladores lógicos programáveis (CLP), até entidades virtuais presentes apenas no ambiente de software. O servidor possui a autonomia para organizar os endereços apropriadamente, podendo também realizar subdivisões no espaçamento para estabelecer hierarquias e estruturas com base nas referências dos nós (Mahnke; Leitner; Dam, 2010).

Outra característica relevante na organização do espaçamento dos nós, é a sua subdivisão em nós dentro ou fora da View, a qual representa um subespaço que se distingue por conter os nós que são visíveis aos clientes conectados ao servidor. Os servidores conseguem estruturar múltiplas Views, estabelecendo uma organização que pode simplificar a descoberta e fornecer ao cliente uma estrutura na qual os nós de cada view estão relacionados entre si (OPC-Foundation, 2021).

Para iniciar a comunicação com os clientes conectados, os servidores empregam uma estrutura em camadas que guarda semelhanças com a dos clientes, como ilustrado na Figura 7. Assim como os clientes, a pilha de comunicação no servidor desempenha a função de converter as chamadas e notificações para a camada de API do servidor. Esta última, por sua vez, interage com a aplicação conforme as necessidades específicas de cada chamada ou resposta a ser realizada (OPC-Foundation, 2021).

Figura 7 – Camadas do servidor OPC-UA



Fonte: Adaptado de (OPC-Foundation, 2021).

A comunicação entre a camada de API e a camada de aplicação pode ocorrer de duas formas distintas: por meio de requisição direta e por meio de subscrição. Na requisição direta, os valores são acessados diretamente pelo espaço de endereçamento dos nós. Já por meio de subscrição, os clientes solicitam a subscrição a um nó específico. Nesse cenário, o servidor, utilizando uma entidade de monitoramento, realiza a notificação para o cliente sempre que ocorre uma mudança nos valores do mesmo, ou quando é ativado um alarme ou evento específico (Mahnke; Leitner; Dam, 2010).

2.2.1.2 OPC-UA PubSub

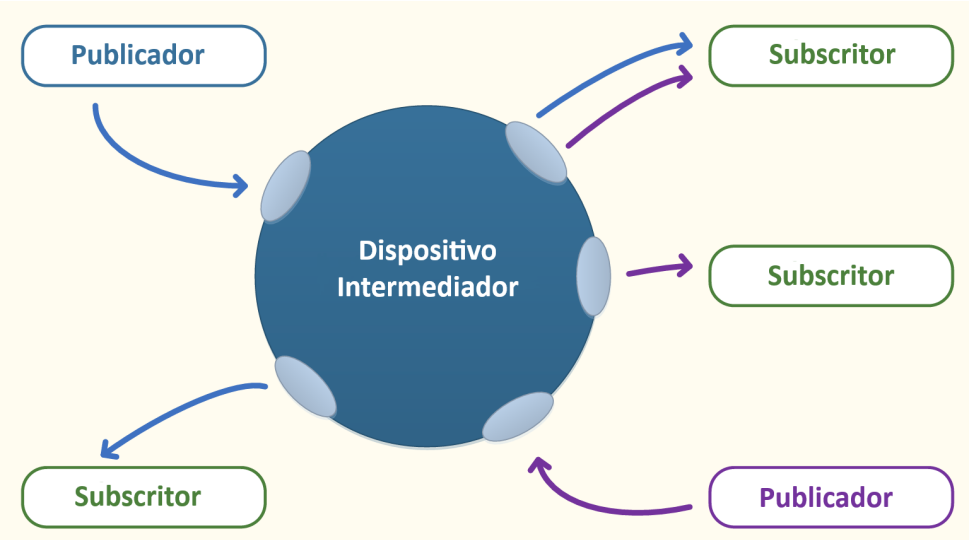
Na especificação 14 da OPC Foundation, foi implementado um novo modelo de comunicação para o protocolo OPC UA, conhecido como modelo de publicação e subscrição. Esse modelo se destaca por proporcionar uma independência entre o cliente e o servidor, pois o protocolo introduz uma camada intermediária de abstração, responsável por gerenciar essa separação entre os comunicadores (OPC-Foundation, 2021).

O desenvolvimento do modelo PubSub foi orientado para ampliar a aplicabilidade do protocolo a cenários nos quais a escalabilidade da comunicação e a independência geográfica são aspectos cruciais a serem considerados. Nesse

modelo, os servidores OPC UA não requerem comunicação direta com os clientes, e os clientes, por sua vez, não precisam ter conhecimento sobre o endereçamento de nós presente nos servidores (OPC-Foundation, 2021).

Essa independência é possibilitada pela introdução de um dispositivo intermediário conhecido como Message Oriented Middleware (MOM). Esse *middleware* atua para facilitar a troca de mensagens de forma assíncrona. Como ilustrado na Figura 8, o MOM atua como uma camada intermediária que permite aos dispositivos comunicarem-se entre si sem a necessidade de estabelecer uma conexão direta, assim publicadores e subscritores interagem independentemente através deste meio de campo (OPC-Foundation, 2021).

Figura 8 – Esquema OPC-UA PubSub



Fonte: Adaptado de (OPC-Foundation, 2021).

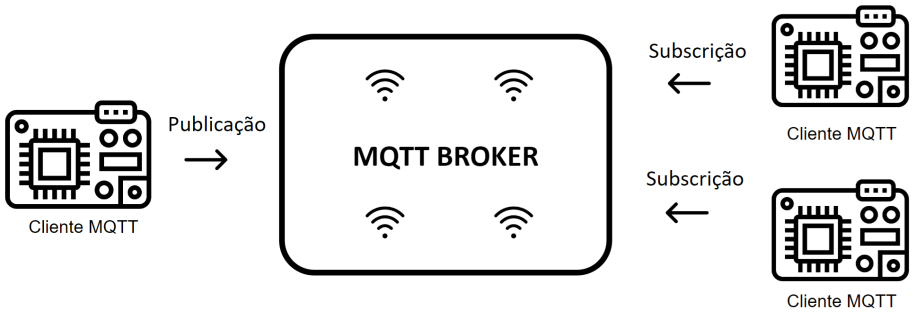
O OPC UA suporta duas variantes principais de dispositivos intermediários: o modelo com e sem a utilização de um broker. No modelo sem um broker, o intermediador é representado pela infraestrutura de rede local, que deve conseguir encaminhar mensagens baseadas em datagrama, utilizando protocolos como o User Datagram Protocol (UDP). Já no modelo baseado no uso de um broker, há a presença de um intermediador local que suporta protocolos de mensagens como o Advanced Message Queuing Protocol (AMQP) ou MQTT. Todas as trocas de mensagens ocorrem através desses protocolos, proporcionando a comunicação entre publicadores e subscritores de forma assíncrona (OPC-Foundation, 2021).

2.3 PROTOCOLO MESSAGE QUEUE TELEMETRY TRANSPORT

A evolução da Indústria 4.0 demanda protocolos de comunicação simples e eficientes, destacando o MQTT nesse contexto (Arif, 2017). Surgindo para atender à necessidade de comunicação assíncrona em redes com largura de banda restrita, o MQTT se alinha com os princípios de simplicidade e assincronismo (Banafa, 2019).

O MQTT tem sua principal caracterização baseada em mecanismos de publicação/assinatura, como mostrado na Figura 9. Nesse mecanismo, dispositivos podem realizar a publicação de determinadas mensagens, e essas mensagens podem ser acessadas por meio da a subscrição ao tópico em que está mensagem foi publicada, por qualquer outro dispositivo com acesso a mesma rede.

Figura 9 – Esquema de Funcionamento do protocolo MQTT



Fonte: Elaborado pelo autor (2023)

Esta concepção, fundamenta-se na autonomia dos clientes MQTT, onde um cliente pode agir sem dependência de outro. Nesse contexto, aqueles que realizam publicações ou subscrições não precisam ter acesso direto ao endereço IP, ou à porta do outro cliente. Todo o intercâmbio de informações ocorre por meio do acesso ao Broker MQTT (Kourtis, 2018).

Essa abordagem oferece uma camada de abstração, que simplifica a interação entre os dispositivos conectados. O Broker MQTT atua como um intermediário, gerenciando as comunicações e garantindo a entrega das mensagens, desta maneira a implementação e o comportamento dos clientes se torna mais simples e eficiente (Banks; Gupta; Hoshino, 2019).

Além disso, a centralização das operações no broker facilita a manutenção e monitoramento do sistema na totalidade, uma vez que é possível ter acesso a todas as trocas de mensagens em um único local. Essa eficiência na troca de informações destaca o MQTT como um protocolo que pode ser facilmente adaptado para a comunicação em ambientes industriais avançados (Sandeep; Thakur, 2018).

2.3.1 Estruturação de um pacote MQTT

Desde sua criação em 1999, o foco do desenvolvimento do protocolo MQTT foi para resolver a necessidade de transmissão de informação em locais com a largura de banda limitada, havendo a necessidade de ser um protocolo leve e eficiente. Para resolver este problema a estrutura do seu cabeçalho deveria ser simples e compacta, nesta subsecção esse será o tema elaborado, visando uma análise da versão cinco do protocolo (Hive-MQ, 2024).

O protocolo MQTT é majoritariamente baseado em pequenos pacotes chamados de pacotes de controle, esses pacotes de controle são os conjuntos de informação que definem qual o sentido da mensagem. O quadro 1 apresenta os 15 diferentes pacotes de controle presentes na atual versão 5.0 do protocolo MQTT (Hive-MQ, 2024).

Quadro 1 – Pacotes de controle MQTT.

Nome	Valor	Direção	Descrição
CONNECT	1	Cliente para Servidor	Requisição de conexão
CONNACK	2	Servidor para Cliente	Reconhecimento de conexão
PUBLISH	3	Ambos	Publicação de mensagem
PUBACK	4	Ambos	Reconhecimento de mensagem (QoS 1)
PUBREC	5	Ambos	Reconhecimento de mensagem (QoS 2)
PUBREL	6	Ambos	Publicação liberada
PUBCOMP	7	Ambos	Publicação completa
SUBSCRIBE	8	Cliente para Servidor	Requisição de subscrição
SUBACK	9	Servidor para Cliente	Reconhecimento de subscrição
UNSUBSCRIBE	10	Cliente para Servidor	Requisição de parar a subscrição
UNSUBACK	11	Servidor para Cliente	Reconhecimento de parar a subscrição
PINGREQ	12	Cliente para Servidor	Requisição de PING
PINGRESP	13	Servidor para Cliente	Resposta do PING
DISCONNECT	14	Ambos	Notificação de desconexão
AUTH	15	Ambos	Troca de autenticação

Fonte: Adaptado de (MQTT, 2024)

Os pacotes de controle representam apenas uma fração do cabeçalho completo

do protocolo MQTT, que, por sua vez, é estruturado em três subcomponentes distintos, cada um com suas próprias subdivisões específicas. A primeira subdivisão de destaque é o cabeçalho fixo, que comporta o tipo de pacote de controle assim como o tamanho total da mensagem. Este cabeçalho é exemplificado no Quadro 2, o qual é considerado obrigatório em todas as mensagens do protocolo, independentemente do tipo de pacote de controle em questão (MQTT, 2024).

Quadro 2 – Cabeçalho fixo MQTT.

Bit	7	6	5	4	3	2	1	0
Byte 1	Tipo de pacote de controle				Flags para cada pacote			
Byte 2...	Resto do cabeçalho							

Fonte: Adaptado de (MQTT, 2024)

O tamanho do cabeçalho fixo varia proporcionalmente ao tamanho total da mensagem, pois a partir do segundo byte deste cabeçalho, é incluído o tamanho total da mensagem a ser transmitida. Adicionalmente, o cabeçalho fixo precede o cabeçalho variável, que pode ser considerado uma informação complementar antes da incorporação da mensagem final. É importante ressaltar que nem todos os pacotes de controle apresentam um cabeçalho variável, assim como o campo da mensagem da aplicação (MQTT, 2024).

Os elementos contidos no cabeçalho variável variam conforme a finalidade do pacote MQTT em questão. No caso do pacote CONNECT, seu cabeçalho variável inclui o nome e o nível do protocolo, as flags de conexão, indicadores que determinam se o cliente permanecerá conectado, além de propriedades adicionais relevantes. Por outro lado, no pacote PUBLISH, o cabeçalho variável é mais simplificado, contendo apenas o nome do tópico a ser publicado, o identificador de pacote e, por fim, suas respectivas propriedades (MQTT, 2024).

Por fim, na estrutura da mensagem MQTT, encontra-se o cabeçalho do payload, que consiste na mensagem da aplicação. No caso do pacote PUBLISH, o cabeçalho contém as informações da mensagem da aplicação destinadas a serem transmitidas para o broker. Este elemento do cabeçalho, embora essencial para a comunicação entre os dispositivos, é exclusivo de cada mensagem e não é padronizado como os cabeçalhos fixo e de controle, uma vez que sua estrutura e conteúdo são determinados pela aplicação que está enviando a mensagem (MQTT, 2024).

2.3.2 Níveis de QoS no MQTT

Na troca de mensagens, é aplicado os níveis de qualidade de serviço (QoS), que se evidencia como alicerce fundamental para a confiabilidade e entrega eficaz de mensagens (Kourtis, 2018). A autenticação e evidência da entrega de cada uma

das mensagens varia conforme a qualidade de serviço aplicada na mesma, a qual pode ter valores de zero a dois, onde cada um desses valores apresenta uma diferente abordagem quanto a entrega da telemetria.

A relação entre os níveis de QoS e as interações entre publicadores e subscritores de mensagens é uma variável fundamental, cuja compreensão é crucial para o desenvolvimento e a implementação de sistemas de comunicação (Kourtis, 2018). O Quadro 3 apresenta de maneira sistemática os diferentes comportamentos associados a cada nível de QoS, elucidando as garantias de entrega e a frequência com que uma mensagem pode ser recebida.

Quadro 3 – Relação de QoS com Pub/Sub.

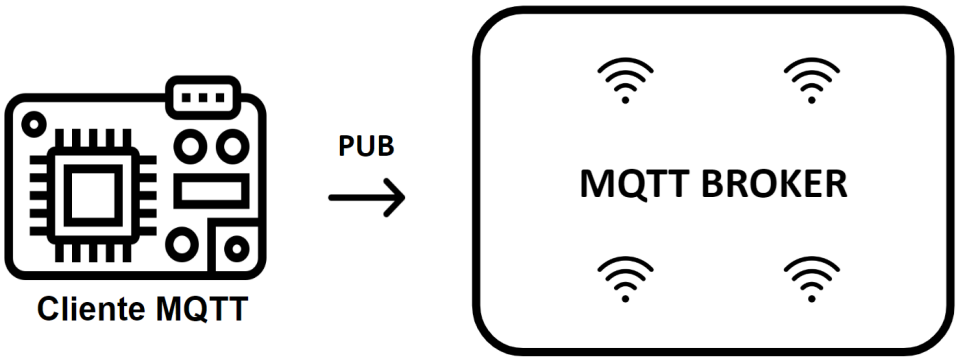
QoS	Publicador	Subscritor
0	Envia a mensagem apenas uma vez	Pode ou não receber a mensagem
1	Envia a mensagem pelo menos uma vez	Pode receber a mensagem uma vez, assim como pode receber mais vezes
2	Envia a mensagem pelo menos uma vez	Receberá a mensagem apenas uma vez

Fonte: Autoria própria (2024).

Embora a garantia de entrega de cada mensagem melhore com o aumento do QoS, essa mudança também traz consigo uma desvantagem, o aumento na quantidade de trocas de mensagens, resultando em uma maior utilização da largura de banda. Isso ocorre porque, à medida que se prioriza a qualidade do serviço, há uma tendência de aumentar a frequência das verificações e confirmações de entrega, o que, por sua vez, gera mais tráfego na rede (Hive-MQ, 2024).

O QoS zero, embora assegure o envio de uma mensagem, não garante sua recepção pelo broker, uma vez que o remetente não aguarda confirmação e descarta a mensagem do armazenamento. Assim, o dispositivo emissário fica desprovido de qualquer informação sobre o estado da mensagem após sua transmissão, resultando em uma ausência de conhecimento quanto à efetividade da entrega. A Figura 10 demonstra que esta lacuna de informação impede a ciência do remetente acerca do sucesso ou fracasso na realização da entrega da mensagem (Hive-MQ, 2024).

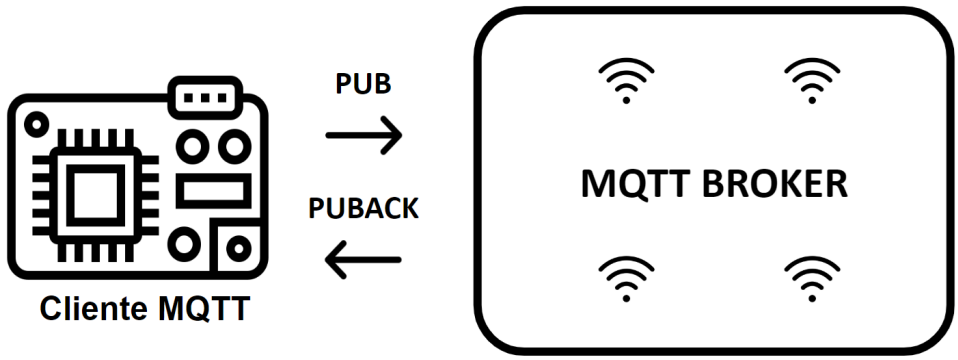
Figura 10 – QoS 0 no protocolo MQTT



Fonte: Elaborado pelo autor (2024)

Durante a execução de um envio de mensagem com um nível de QoS definido como um, ocorre uma troca de mensagens que habilita o dispositivo responsável pela publicação da mensagem a receber uma confirmação do broker, ratificando a chegada bem sucedida da mensagem. Neste meio tempo a mensagem é guardada até o recebimento da confirmação. A Figura 11 ilustra o sequenciamento dessas mensagens, evidenciando o envio de um pacote PUB pelo cliente seguido pelo recebimento do reconhecimento PUBACK do broker, validando assim o registro da mensagem (HiveMQ, 2024).

Figura 11 – QoS 1 no protocolo MQTT



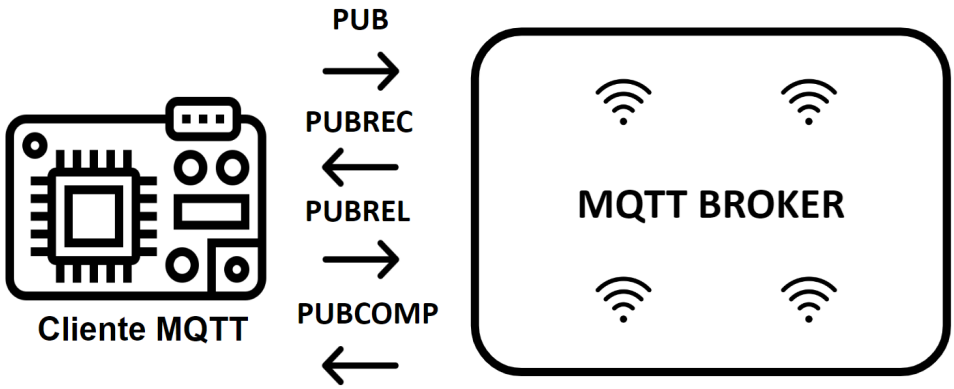
Fonte: Elaborado pelo autor (2024)

Nesse contexto, quando uma mensagem é enviada por um dispositivo utilizando o nível de QoS um, e o reconhecimento de mensagem (ACK) esperado do broker não é recebido pelo publicador em um intervalo de tempo predefinido, o remetente reenvia a mensagem. No entanto, sob este modelo de QoS, mesmo que a mensagem tenha sido entregue com sucesso, é possível que o pacote PUBACK não tenha sido recebido pelo dispositivo emissor no prazo estabelecido, ou até mesmo não tenha sido retornado. Neste cenário, a mensagem é reenviada, resultando na possibilidade de o dispositivo

receptor receber a mesma mensagem repetidamente (MQTT, 2024).

Diferentemente dos serviços de qualidade zero e um, o QoS dois apresenta uma estrutura mais complexa, envolvendo múltiplos handshakes, visando garantir a entrega exata de uma única mensagem ao broker destinatário. A Figura 12 ilustra que o inicio da sequência de trocas de pacotes utilizada no nível dois do QoS, se inicia com o envio do pacote PUB, seguido por uma confirmação com o pacote PUBREC. Subsequentemente, ocorre a troca de dois outros pacotes, o PUBREL e o PUBCOMP, completando assim o ciclo de garantia de entrega da mensagem (MQTT, 2024).

Figura 12 – QoS 2 no protocolo MQTT



Fonte: Elaborado pelo autor (2024)

Neste ciclo de comunicação, uma vez que o cliente transmissor recebe a confirmação de recebimento PUBREC após enviar o pacote PUB, prossegue então ao envio do pacote PUBREL, que sinaliza ao destinatário que a mensagem foi devidamente processada, permitindo assim sua remoção dos buffers temporários. Por conseguinte, o pacote PUBCOMP é gerado e encaminhado de volta ao dispositivo emissor, confirmando a conclusão bem-sucedida de toda a transmissão da mensagem (MQTT, 2024).

Durante o envio de cada uma das mensagens é feito o uso de identificadores de mensagens, esta abordagem facilita o monitoramento do fluxo de mensagens e a resolução de eventuais falhas de comunicação. Tanto na implementação do nível de serviço de qualidade um, quanto no nível dois, estes identificadores atuam como meios de garantia para o tratamento de múltiplas mensagens, e múltiplos dispositivos, visto que o MQTT é um protocolo que suporta o envio de mensagens em paralelo (MQTT, 2024).

No âmbito da segurança, durante o envio dos pacotes, o MQTT aborda essa preocupação por meio de protocolos como TLS/SSL, que garante a integridade e confidencialidade das informações transmitidas. Essa abordagem, faz com que o MQTT seja escolhido em diversos ambientes, especialmente em ambientes industriais, onde

a proteção dos dados é inegociável (Sandeep; Thakur, 2018).

2.4 COMPUTAÇÃO DE BORDA

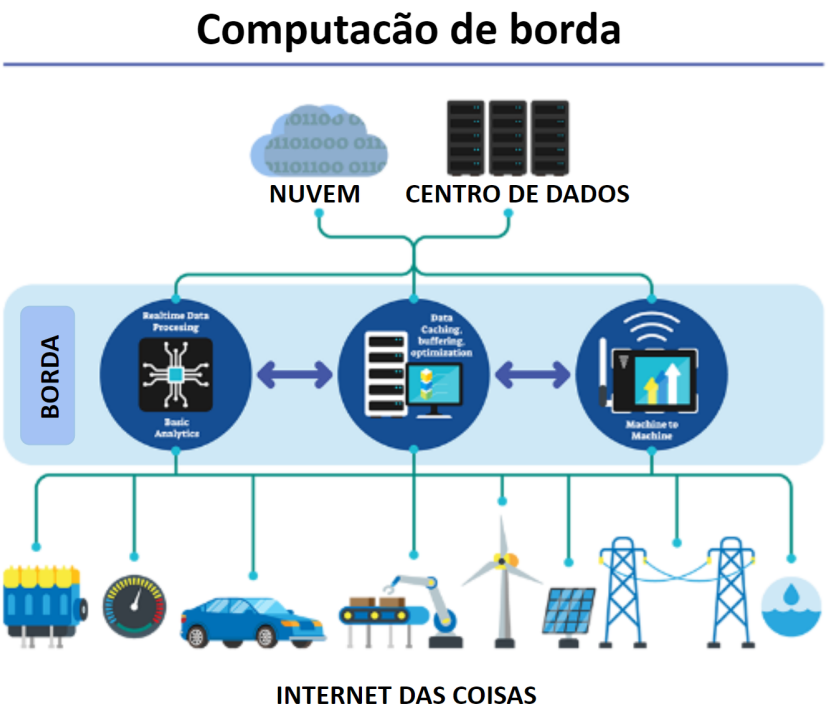
Segundo Parashar et al. (2016), a evolução da tecnologia em ambientes industriais desencadeou um movimento de crescimento exponencial na geração de dados. Em 2020, estimou-se existirem aproximadamente 50 bilhões de dispositivos conectados à internet. Esse cenário implica em um aumento significativo no tráfego de dados, assim se constrói um desafio na transferência dessas informações dos dispositivos para servidores na nuvem, onde serão processadas.

Com isso, a evolução da computação de borda emerge como um componente essencial no contexto da Indústria 4.0, onde a demanda por processamento de dados próximo às fontes de geração torna-se imprescindível. A computação de borda é um meio de computação, a qual ocorre da borda da rede, e os dados neles computados são provenientes de duas direções opostas: dispositivos de internet das coisas e computação em nuvem. (Cao *et al.*, 2020)

A computação de borda consiste na realização de processamento e análise de dados próximo às origens de produção, minimizando a necessidade de transferência de grandes volumes de dados para nuvens distantes. Essa abordagem descentralizada visa superar desafios como latência, largura de banda e segurança. Ela proporciona uma infraestrutura local para a implementação de sistemas que podem ser usados para tratar dos dados provenientes dos dispositivos IIoT (Qiu *et al.*, 2020).

A Figura 13 ilustra casos de uso da Computação na Borda, onde a nuvem ocupa um extremo e os dispositivos da internet das coisas (IoT), como veículos inteligentes, sensores de rede de energia e atuadores industriais, estão no outro extremo. Entre as aplicações IoT e os centro de dados na nuvem, os dispositivos de borda desempenham o processamento e a comunicação de dados nesse espaço intermediário (Rios, Porto Alegre, 2022).

Figura 13 – Computação na borda



Fonte: IEEE (2022, p. 7).

Uma das características da aplicação da computação de borda, é o seu papel de intermediador entre os dispositivos IIoT e os terminais na nuvem, garantindo o isolamento entre esses extremos. Essa arquitetura de isolamento na computação de borda é comumente organizada em três camadas distintas: a camada de terminais, a camada de barreira e a camada de nuvem (Cao *et al.*, 2020).

A camada de terminais engloba todos os dispositivos conectados à borda da rede local, como sensores e computadores. Nessa camada, os dispositivos IIoT não são meramente consumidores de dados, mas também fornecedores ativos de informações. Por outro lado, a camada de nuvem é composta por servidores e dispositivos de armazenamento, é nessa camada que se torna viável executar operações computacionais que excedem as capacidades das demais camadas da infraestrutura (Cao *et al.*, 2020).

A camada de barreira é a responsável pela interligação das duas outras camadas da infraestrutura. Composta por pontos de acesso, switches e aplicações de gateways de comunicação, essa camada serve como um intermediário para facilitar a comunicação entre os dispositivos da camada de terminais. Sua localização na borda da rede proporciona uma diminuição na latência da comunicação entre os dispositivos terminais, quando comparado a comunicação com a nuvem (Cao *et al.*, 2020).

Além de atuar na preservação da largura de banda, com o processamento local, a segurança dos dados é melhorada, uma vez que a proximidade física com os

dispositivos permite a implementação de medidas de proteção. A aplicação de medidas de proteção a dados sensíveis é facilitada pela possibilidade de desconexão com sistemas externos, ou pela implementação de modelos de comunicação que respeitam determinada estrutura (Shi; Cao; Zhang, 2016).

2.5 TRABALHOS RELACIONADOS

O trabalho proposto por Luchian *et al*. (2021) se assemelha fortemente ao trabalho proposto. Com uma metodologia sistemática, o trabalho propõe uma computação de ponte onde é feita a integração de informações enviadas por protocolos MQTT e OPC-UA em um único meio. Contudo, fora uma análise de dados estatísticos sobre a eficiência de cada abordagem, o mesmo não assegura a integridade dos dados recebidos pelo concentrador.

Visando uma aplicação de Reconfigurable Manufacturing System (RMS), o trabalho proposto por Liu e Bellot (2020) apresenta uma solução modularizada para a integração de dados MQTT e OPC-UA. A implementação de tal conceito baseia-se na aplicação de um sistema de publicação e subscrição. Contudo, a comunicação realizada apenas serve como um intermédio de comunicação entre os aparelhos já configurados, impossibilitando no estado atual de desenvolvimento uma integração com um banco de dados.

Um trabalho similar com conectividade a nuvem foi realizado por Bosi *et al*. (2020), onde foi realizada a integração de protocolos de comunicações de diferentes arquiteturas. Com os dados já coletados, houve a implementação a visualização dos mesmos. Apesar de toda a similaridade, as diferenças se apresentam na utilização de diferentes protocolos de comunicação, onde o trabalho realizado por Bosi apresenta um foco nos protocolos MQTT e AMQP.

Park, Kim e Kim (2018) apresentaram uma solução de integração do envio de dados de múltiplos dispositivos IoT. No trabalho é apresentado uma arquitetura de múltiplas camadas, com a análise de diferentes variações do protocolo MQTT. Contudo, a integração desses dispositivos é somente baseada na comunicação MQTT, portanto não suporta o protocolo OPC-UA.

3 MATERIAIS E MÉTODOS

A metodologia adotada neste trabalho envolve a aplicação de métodos específicos para coleta, análise e interpretação de dados, assim como para a implementação dos códigos necessários, buscando garantir a confiabilidade e a validade das conclusões alcançadas. A seguir, serão detalhadas as etapas metodológicas, destacando os procedimentos utilizados para cada fase do estudo.

Para a implementação do projeto proposto neste trabalho, inicialmente é preciso definir claramente o ideal de funcionamento do projeto, o qual será definido pelo conceito de operação do mesmo. Após isso, requisitos funcionais e não funcionais serão detalhadamente especificados, seguidos pela modelagem e planejamento para a construção do demonstrador.

Um dos focos da estruturação desse trabalho é permitir a viabilização de futuras montagens e expansões do projeto. Com isso será apresentado o fluxo de dados, o mapeamento dos processos e posteriormente o cenário de implementação, proporcionando uma visão abrangente das etapas a serem realizadas.

3.1 CONCEITO DE OPERAÇÃO

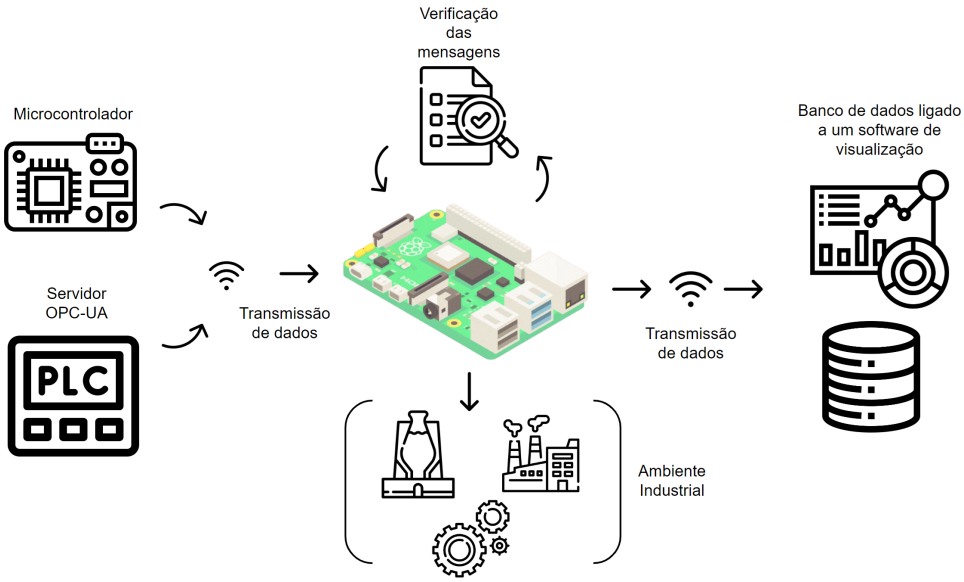
O conceito de operação, em um contexto técnico, denota a articulação das atividades, procedimentos e interações que delineiam o funcionamento de um sistema ou projeto. Esse conceito, baseia-se na análise desde as operações fundamentais até os processos mais intricados, visando garantir a eficiência, segurança e eficácia do sistema (Pressman, 2014).

No âmbito de projetos e sistemas, o conceito de operação orienta o desenvolvimento, implementação e manutenção contínua. Sua abrangência envolve aspectos como interação com usuários, integração com outros sistemas, manipulação de dados, resposta a eventos específicos e gestão de falhas potenciais.

Um conceito de operação bem delineado oferece uma visão completa do funcionamento do sistema em diversas situações, assim, facilita a tomada de decisões, identificação de requisitos adicionais e avaliação do desempenho em cenários práticos (Pressman, 2014). Neste viés, na Figura 14 é apresentado um conceito de operação de funcionalidade geral do sistema proposto neste trabalho.

É possível identificar, o ideal operacional, onde dados serão enviados de microcontroladores e controladores lógicos programáveis (CLPs), para um computador de borda presente no centro da imagem. Por meio da identificação do conteúdo recebido e verificação da integração dos dados, estes são validados para serem enviados para um banco de dados com integração de um software de visualização.

Figura 14 – Conceito de operação do sistema



Fonte: Elaborado pelo Autor (2023)

3.2 REQUISITOS DE PROJETO

A definição detalhada dos requisitos funcionais e não funcionais no início do projeto desempenha um papel crucial na orientação das etapas subsequentes. Ao identificar claramente as funcionalidades necessárias, cria-se um guia para a modelagem e o planejamento, garantindo que o desenvolvimento do demonstrador atenda às expectativas e necessidades específicas (Conjo, Quixadá, 2022).

3.2.1 Requisitos funcionais

Os requisitos funcionais mapeiam as funcionalidades atendidas pelo sistema por meio de suas funções ou serviços. Ao estabelecer respostas precisas para entradas tanto conhecidas quanto desconhecidas, esses requisitos conferem confiabilidade às aplicações (Conjo, Quixadá, 2022).

Na sequência, serão apresentados os requisitos funcionais para o sistema. Esses requisitos funcionais serão subdivididos conforme suas características de funcionamento.

Coleta de Dados:

[RF-001] Recepção de Dados de Dispositivos IIoT.

O sistema deve poder receber dados provenientes de dispositivos IIoT, garantindo a interoperabilidade e a capacidade de integração com diferentes fontes de dados industriais.

Processamento Local:

[RF-002] Capacidade de Processamento Local.

O sistema deve conseguir realizar o processamento local dos dados, possibilitando análises preliminares diretamente nos dispositivos para otimizar a eficiência do sistema. Preliminares, as quais, estão neste escopo, limitadas a verificação da estrutura dos dados recebidos e a integridade do mesmo.

[RF-003] Algoritmos de isolamento de clientes.

Deve implementar algoritmos específicos capazes de lidar com dados em tempo de execução, sem necessidade de postergar um processo de um cliente devido ao recebimento de novos dados.

Protocolos de Comunicação:

[RF-004] Suporte ao protocolo MQTT.

O sistema deve oferecer suporte ao protocolo de comunicação MQTTv5.

[RF-005] Suporte ao protocolo OPC-UA.

O sistema deve oferecer suporte ao protocolo de comunicação OPC-UA, em seu modelo PubSub.

Perda de dados:

[RF-006] Armazenamento Local.

Capacidade de armazenar dados localmente na máquina utilizada, garantindo uma abordagem descentralizada e a disponibilidade de dados mesmo em condições de conectividade intermitente.

[RF-007] Integração com Banco de Dados.

Deve permitir a integração com um banco de dados para armazenamento, possibilitando o acesso remoto e a análise centralizada dos dados coletados.

Visualização:

[RF-008] Conectividade com um software de visualização.

O sistema deve ter o poder de integrar os dados recebidos pelo banco de dados com algum software de visualização.

Conectividade e Escalabilidade:

[RF-009] Manutenção de Conectividade com Múltiplos Dispositivos.

Deve manter conectividade com múltiplos dispositivos simultaneamente, garantindo a eficiência operacional mesmo em ambientes com alta densidade de dispositivos.

3.2.2 Requisitos não funcionais

Os requisitos não funcionais constituem-se de diretrizes para alcançar as funcionalidades desejadas do sistema, estes requisitos são intrinsecamente ligados à arquitetura do sistema, desempenhando um papel crucial na garantia de sua qualidade e desempenho. Desta maneira, eles transcendem simples funcionalidades, abordando aspectos que permeiam a eficiência, segurança, usabilidade, disponibilidade, compatibilidade e manutenibilidade do sistema (Conjo, Quixadá, 2022).

A seguir, seguem os requisitos não funcionais que orientaram o desenvolvimento e implementação do sistema, assegurando que este atenda não apenas às expectativas funcionais, mas também aos mais elevados padrões de desempenho e qualidade.

Desempenho:
[RNF-001] Tempo de Resposta.
O sistema deve garantir um desempenho com tempos de resposta em até 5 segundos.

Segurança:
[RNF-002] Protocolos de Segurança de Perda de Dados.
Implementação de protocolos de segurança como QoS1 e CRC para assegurar a integridade e confiabilidade das informações transmitidas.
[RNF-003] Garantir a integridade e confidencialidade das informações durante todo o ciclo de vida.

Usabilidade:
[RNF-004] Modularidade.
O sistema deve apresentar uma interface modular para facilitar o uso por parte dos usuários, promovendo uma interface intercambiável conforme a necessidade de cada dispositivos IIoT.
[RNF-005] Suporte a Documentação.
Fornecer suporte a documentação clara e recursos de ajuda, contribuindo para a compreensão do sistema e o treinamento eficiente dos usuários.

Disponibilidade:
[RNF-006] Taxa de Disponibilidade.
Manter uma taxa de disponibilidade acima de 90% do tempo em que tiver em execução, durante um dia.
[RNF-007] Estratégias de Recuperação de Falhas.
Implementação de estratégias de recuperação de falhas para garantir a continuidade

operacional em casos de incidentes.

Compatibilidade:
[RNF-008] Compatibilidade com Dispositivos e Sistemas Operacionais.
Assegurar a compatibilidade do sistema com diferentes dispositivos e sistemas operacionais, garantindo uma ampla interoperabilidade.

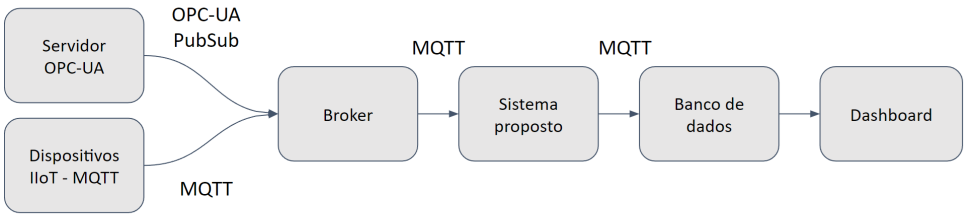
Testabilidade:
[RNF-009] Testes Unitários.
Desenvolver testes unitários extensivos para garantir a interoperabilidade do sistema, e a correta funcionalidade do mesmo.

Manutenibilidade:
[RNF-010] Facilidade de Manutenção.
Facilitar a manutenção do sistema mediante uma arquitetura modular para cadastro de novos dispositivos IIoT e um código bem documentado, simplificando intervenções e atualizações.

3.3 GERENCIAMENTO DE FLUXO

A garantia do fluxo de dados dos dispositivos IIoT para o banco de dados é o foco principal da metodologia adotada neste trabalho. Um diagrama de sequência dos dados é apresentado na Figura 15, assim como o protocolo de comunicação utilizado em cada uma das etapas de trocas de mensagens.

Figura 15 – Diagrama do fluxo de dados



Fonte: Elaborado pelo Autor (2024)

Com o propósito de satisfazer o requisito funcional **RF-001**, a entrada primária para o sistema é o serviço de mensagens MQTT fornecido pelo broker Mosquitto (Mosquitto, 2024). Esta escolha de broker MQTT foi realizada devido à sua configuração simplificada e à capacidade de suportar a entrada de dados por meio dos protocolos

MQTT e OPC-UA em seu modelo PubSub.

A partir da disponibilidade dos dados recebidos pelo broker, os mesmos são capturados para o sistema proposto por meio da utilização de uma biblioteca fundamentada na implementação Paho-mqtt (Paho-Mqtt, 2024). Durante esta fase, o cliente MQTT, encarregado da subscrição, estabelece conexões com tópicos específicos no broker e assim que uma mensagem é recebida, ela é direcionada para o sistema.

Na fase de saída do projeto proposto, os dados são encaminhados para um banco de dados mediante a operação de um cliente MQTT, cuja implementação é baseada no cliente Paho novamente. Assim, tanto a entrada quanto a saída de dados do sistema são efetuadas por meio do mesmo protocolo, desse modo o software se comporta de maneira transparente a entradas e saídas, e atua como um filtro.

Como interface de visualização dos dados, é definido a plataforma de código aberto para dispositivos IoT Thingsboard (Thingsboard, 2024). Esta plataforma conta com um sistema de manejo de dispositivos, baseado em tokens de acesso, em que, cada dispositivo apenas consegue se comunicar com a plataforma caso tenha acesso ao token gerado pela mesma no cadastro do dispositivo IoT.

A plataforma também é integrada com diversos tipos de banco de dados, sendo o escolhido para o sistema em questão, o PostgreSQL, cuja integração é feita no momento da instalação do Thingsboard. A escolha desta plataforma parte dos requisitos funcionais **RF-007** (Integração com Banco de Dados) e **RF-008** (Integração com software de visualização) os requisitos não funcionais **RFN-004** (Modularidade) e **RFN-010** (Facilidade de manutenção).

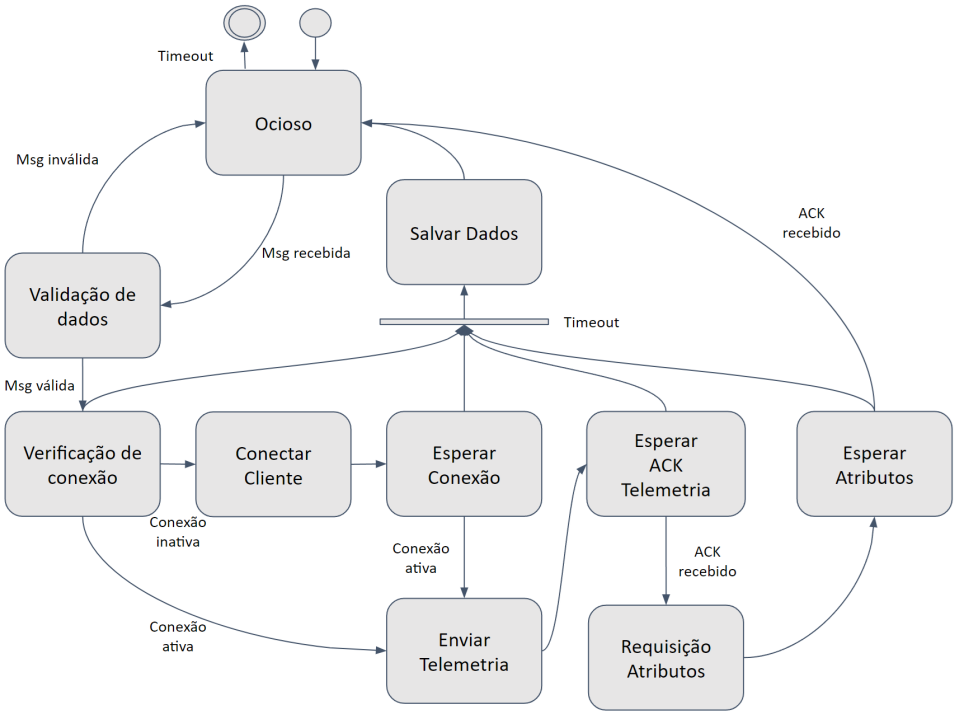
3.4 MODELAGEM

A partir da análise do conceito operacional do projeto e da conceituação do fluxo de dados, foi desenvolvido um diagrama de estados de um consumidor para o tratamento das mensagens, conforme ilustrado na Figura 16. Esse diagrama foi concebido com base no **RF-006**, sobre a necessidade de armazenamento local de mensagens com o intuito de garantir a entrega mesmo em caso de falha de comunicação.

Neste diagrama, tanto o início quanto o fim do sistema são representados pelo estado Ocioso, controlado por uma fila. Se nenhuma nova mensagem for recebida num determinado período, o consumidor entra em seu estado de finalização. No entanto, se uma mensagem for adicionada à fila, os estados do consumidor começam a ser alterados conforme o processamento da mensagem é realizado.

No estado de validação de dados, o consumidor verifica determinados campos presentes na estrutura da mensagem, detalhes na seção Estruturação da mensagem.

Figura 16 – Diagrama consumidor



Fonte: Elaborado pelo Autor (2024)

Caso algum campo essencial não esteja presente, o consumidor adiciona caso seja possível. Vale ressaltar que uma pré-verificação já é realizada na mensagem antes da mesma chegar ao consumidor.

No estágio de verificação da conectividade, se a conexão com o banco de dados já estiver estabelecida, o próximo passo é enviar a mensagem. Caso contrário, o cliente tentará estabelecer a conexão com o Thingsboard. Após a conexão ser estabelecida, a mensagem será enviada e o consumidor aguardará a confirmação do recebimento da mesma.

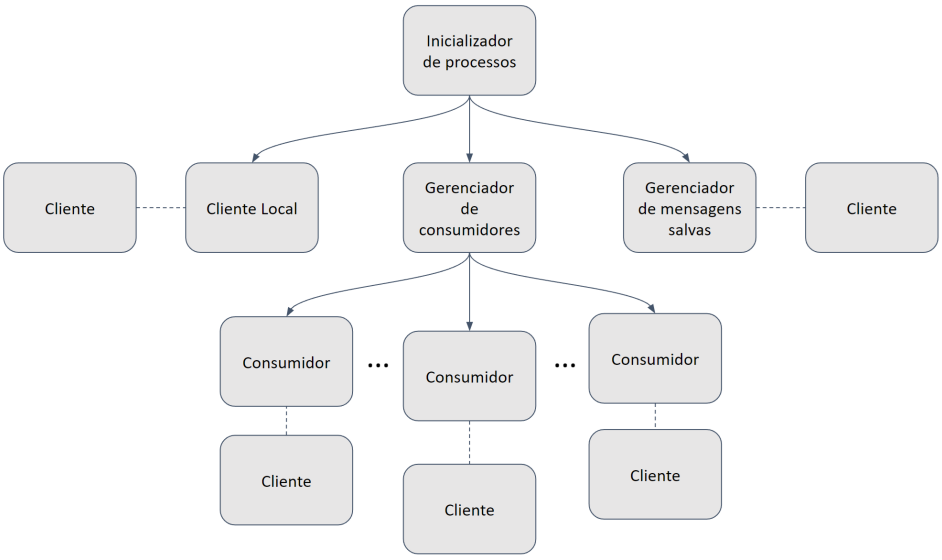
Após a confirmação da telemetria recebida, o processo de troca de mensagens é retomado para a requisição de atributos, com o intuito de fornecer ao dispositivo IoT informações armazenadas no Thingsboard. É importante destacar que há um limite de tempo em estados de espera e verificação, caso esse limite seja excedido, a mensagem será armazenada localmente para posterior envio.

3.4.1 Mapeamento dos processos

Após a elaboração do diagrama de um consumidor, responsável pelo envio de uma mensagem, se fez necessário a definição de um mapa de processos para a visualização do sistema na totalidade. Dessa maneira, está presente na Figura 17, uma visualização superficial dos processos iniciados pelo sistema.

A arquitetura elaborada para este diagrama foi concebida com o propósito de

Figura 17 – Mapeamento dos processos



Fonte: Elaborado pelo Autor (2024)

atender ao requisito funcional **RF-010**. Esse requisito destaca a necessidade de oferecer suporte a múltiplas conexões, uma para cada cliente, o que se alinha diretamente com os objetivos estabelecidos no escopo do trabalho proposto.

Nesta estrutura, identificam-se três componentes principais inicializados como processos pelo laço de execução principal. O primeiro deles é o cliente local, cuja função é coletar as mensagens disponibilizadas em cada um dos tópicos abertos pelo broker Mosquitto. Esse processo de coleta é realizado por meio de um cliente MQTT implementado internamente neste componente.

Outro processo principal é o gerenciador de mensagens armazenadas. Este processo também é composto por outro cliente responsável pela confirmação de conexão com o banco de dados. Uma vez que o cliente interno a este processo confirma a conexão com o servidor, é realizada uma verificação das mensagens salvas no armazenamento local.

As mensagens salvas localmente são então redirecionadas do processo de gerenciamento local para o processo de gerenciamento de consumidores. O processo que cuida dos consumidores se faz presente como o principal componente do programa, uma vez que é responsável por estabelecer a comunicação de receptor entre esses três processos.

Ao receber os dados provenientes dos outros processos, o gerenciador de consumidores inicia um novo processo para cada consumidor, encarregado de realizar o envio das mensagens. Além disso, o gerenciador de consumidores assume a responsabilidade pelo redirecionamento das mensagens para os consumidores que ele inicializa.

O gerenciador de consumidor também impõe uma restrição ao número de

consumidores e em relação à identidade de cada um. Uma vez que, cada consumidor é responsável pela troca de informações entre o banco de dados e apenas um único dispositivo IIoT.

3.5 IMPLEMENTAÇÃO

No que diz respeito à implementação do projeto, foi adotado o ambiente de desenvolvimento integrado (IDE) Visual Studio Code, na versão 1.88.1 (VSCode, 2024). Todo o código do projeto foi elaborado utilizando a linguagem de programação Python, na versão 3.11.8 (Python, 2024).

No que se refere ao ambiente de programação, com o intuito de emular um sistema similar ao Raspbian, optou-se por criar uma máquina virtual baseada no sistema Lubuntu (Lubuntu, 2024), uma variante da popular distribuição Ubuntu. A seleção desta distribuição foi motivada pelo seu baixo consumo de recursos computacionais, tornando-a adequada para execução em ambientes com capacidade de processamento limitada.

Outro aspecto considerado na seleção da distribuição foi que o Lubuntu, por compartilhar a mesma base do sistema operacional Ubuntu, oferece suporte nativo à plataforma Thingsboard. Além disso, para simular a capacidade computacional de um Raspberry Pi, os recursos da máquina virtual foram restritos a 4 GB de memória RAM e 4 Threads de processamento.

A plataforma Thingsboard foi implantada por meio de um contêiner criado utilizando a tecnologia de código aberto Docker. Essa abordagem de implantação confina a operação da plataforma ao ambiente em que está sendo executada, proporcionando isolamento ao código. Além disso, essa metodologia oferece portabilidade, permitindo que o sistema seja facilmente transferido para diferentes computadores e sistemas operacionais sem a necessidade de configurações adicionais.

No que diz respeito à implementação do código, optou-se pela utilização de programação orientada a objetos, visando promover o encapsulamento e a modularidade de cada uma das classes. Cada componente identificado no mapeamento dos processos foi convertido em uma classe distinta, cujo comportamento é derivado da descrição abrangente do processo em questão. Essa decisão foi pautada nos requisitos funcionais **RF-009** a fim de possibilitar futuras melhorias no software.

Além das bibliotecas implementadas por padrão na versão utilizada do Python, foi utilizada a biblioteca Paho-mqtt como base para a implementação do código do cliente MQTT utilizado no projeto. Outra biblioteca utilizada foi a Jsonschema, que possibilitou a validação de formatação de arquivos JSON de maneira mais eficiente, e a biblioteca Mypi foi responsável por auxiliar a execução da construção e manutenibilidade do código, por meio da especificação de tipos na construção do sistema.

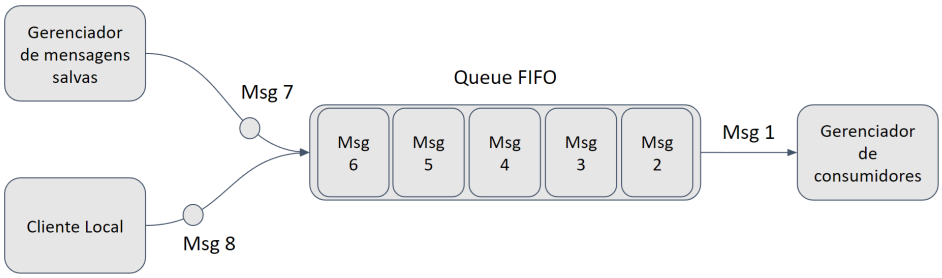
3.5.1 Comunicação entre processos

Cada um dos processos inicializados foi implementado de maneira a garantir uma comunicação assíncrona entre os diferentes componentes do código. Para isto foi feito o uso do conceito de filas, implementadas com o algoritmo First In First Out (FIFO). Essa abordagem proporciona uma organização estruturada das mensagens e dados compartilhados entre os processos, garantindo a consistência e a integridade das operações realizadas pelo sistema.

Essas filas são responsáveis pelo gerenciamento da troca de mensagens, garantindo que a primeira mensagem a ser recebida por um processo seja a primeira a ser tratada. Devido à alocação dinâmica de consumidores no código, a quantidade de filas varia conforme a necessidade de execução, embora sua aplicação seja delimitada em dois casos de uso específicos.

O primeiro caso de uso é caracterizado pela interação de três processos distintos: o cliente local, o gerenciador de mensagens armazenadas e o gerenciador de consumidores. Nesse cenário, o cliente local e o gerenciador de mensagens armazenadas desempenham o papel de produtores, responsáveis por fornecer dados. Por outro lado, o gerenciador de consumidores atua como um consumidor, processando as informações recebidas, o esquema de transferência é encontrado na Figura 18.

Figura 18 – Esquema FIFO entrada



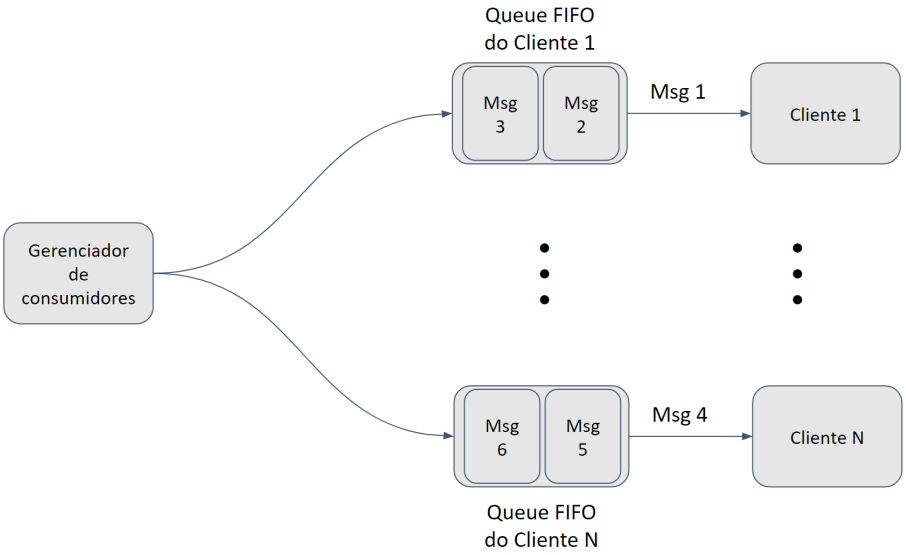
Fonte: Elaborado pelo Autor (2024)

No segundo caso de uso, a dinâmica se concentra na interação entre o gerenciador de consumidores e os próprios consumidores. Nesta etapa da aplicação ocorre um redirecionamento da comunicação para os consumidores com base nos dispositivos IIoT que eles representam. É importante ressaltar que cada consumidor está associado a um único token de acesso, ou seja, está designado a enviar e receber informações específicas de um único dispositivo IIoT.

Ao inicializar cada consumidor, o gerenciador de consumidores atribui a cada um uma fila exclusiva. Assim, quando uma nova mensagem chega ao gerenciador, ela é encaminhada para o consumidor correspondente, identificado pelo token de acesso contido em cada mensagem. Assim, é garantido que cada dispositivo IIoT seja atendido

individualmente por um único consumidor, como apresentado na Figura 19.

Figura 19 – Esquema FIFO consumidores



Fonte: Elaborado pelo Autor (2024)

Entretanto, um limite foi estabelecido para a quantidade máxima de consumidores que podem estar ativos simultaneamente, por meio de um parâmetro configurável pelo usuário. Essa restrição visa evitar vazamentos de memória em ambientes de computação de borda, onde os recursos são limitados. É importante destacar que, após a conclusão de suas tarefas, cada consumidor é encerrado pelo gerenciador após um período de espera específico, também ajustável conforme a necessidade do usuário.

3.5.2 Estruturação da mensagem

Durante a transmissão de dados, outras mensagens não desejadas podem estar presentes no meio constituinte da rede. Dessa maneira foi uma abordagem para a verificação da estruturação da mensagem recebida no gerenciador de consumidores.

As mensagens recebidas são encaminhadas aos respectivos consumidores somente se estiverem formatadas corretamente. Para garantir isso, o formato de transmissão de dados foi padronizado para o formato JSON. Caso a mensagem chegue em um formato diferente o gerenciador de consumidores apenas descarta a mensagem.

Na estrutura da mensagem, o gerenciador também desempenha o papel de verificar os campos obrigatórios. Estes campos são essenciais para garantir que os consumidores enviem as mensagens de maneira adequada. A Figura 20 apresenta um modelo que impõe restrições às variáveis, verificando não apenas a integridade dos

dados, mas também os tipos de cada campo, assegurando a consistência e a validade das informações transmitidas.

Figura 20 – Esquema estrutura JSON

```
schema = {

    "type": "object",

    "properties": {
        "MessageId": {"type": "string"},
        "MessageType": {"type": "string"},
        "PublisherId": {"type": "string"},
        "DeviceToken": {"type": "string"},
        "Messages":    {"type": "object"}},

    "required": ["PublisherId", "DeviceToken", "Messages"]

}
```

Fonte: Elaborado pelo Autor (2024)

Dentro do campo mensagens, a informação apenas deverá respeitar a estrutura de um objeto JSON. Isso se alinha com o propósito do software, que visa operar de forma transparente em relação aos dados transmitidos, garantindo apenas a sua entrega. Se a mensagem estiver em conformidade com os parâmetros estabelecidos, uma etapa adicional de verificação é conduzida. No entanto, essa verificação é realizada no loop principal do consumidor.

O consumidor, ao realizar a análise de estrutura da mensagem, verifica pela presença de um campo contendo a informação com a data e hora de que a mensagem foi enviada. Caso essa informação não esteja presente, o mesmo realiza a inserção dessa informação com a data e hora local presente no dispositivo de computação de borda que está realizando a execução do software. Essa metodologia é empregada com o objetivo de não realizar a exclusão de dispositivos que não possuam um módulo temporizador.

4 RESULTADOS E DISCUSSÕES

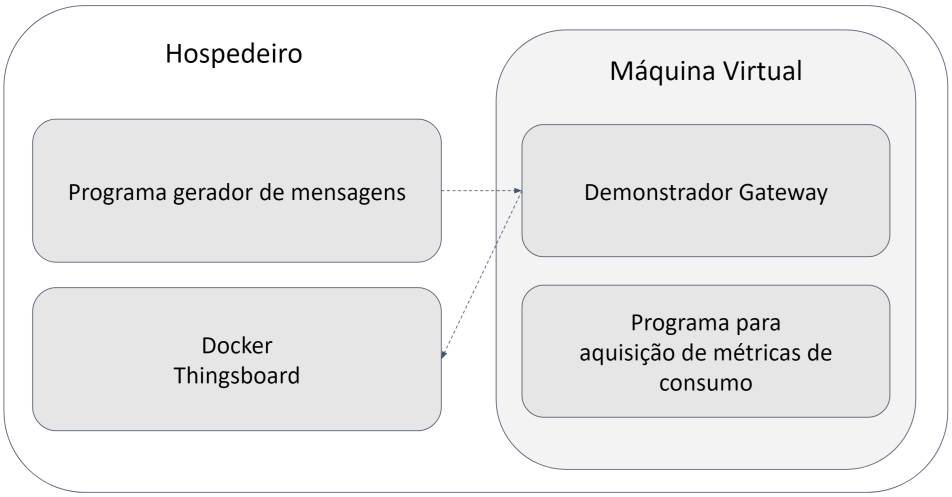
Este capítulo apresenta os resultados alcançados na implementação do *gateway* de comunicação MQTT e OPC-UA. Inicialmente, serão detalhados os cenários adotados para as validações, assim como as configurações aplicadas a cada teste. Em seguida, serão discutidos os resultados dos testes realizados para cada cenário. Por fim, serão abordadas as limitações identificadas no sistema e conduzida uma avaliação em relação aos requisitos especificados na seção de materiais e métodos.

4.1 TESTES

Para validar a implementação do demonstrador do *gateway* de comunicação MQTT e OPC UA, foram delineados três cenários fundamentais para análise. Os cenários foram implementados em um ambiente único, caracterizado pelo sistema operacional Linux Lubuntu, equipado com 4 GB de memória RAM e 2 *cores* de processamento. Essa escolha foi motivada pelas considerações detalhadas na seção de implementação no capítulo de Materiais e Métodos.

Para obter métricas de desempenho, foi necessário isolar o software em questão, movendo o contêiner com a plataforma *ThingsBoard* para o computador hospedeiro da máquina virtual. Na Figura 21 é demonstrado a abordagem adotada com o intuito de garantir a fidelidade dos resultados, uma vez que o demonstrador foi projetado para operar em ambientes com recursos computacionais limitados. Durante a execução dos testes, foram coletadas métricas relacionadas ao uso de processamento e consumo de memória.

Figura 21 – Esquema de isolação VM - Hospedeiro



Fonte: Elaborado pelo Autor (2024)

Uma das análises aplicadas, foi o tempo total de latência que o demonstrador acrescenta em um suposto sistema. Nessa etapa, foram coletadas informações com relação à latência relacionada a troca de informações por conta de cada processo, ou seja, a latência de fila. Também foram coletados dados como a latência total, e a latência para o recebimento do ACK, visto que a mesma inevitavelmente influência na execução do programa.

Além das métricas relacionadas ao hardware, foi realizada uma verificação de perda de mensagens na execução do software, a fim de verificar a eficiência da implementação quando comparada ao envio de mensagem apenas com o uso de QoS 1. Foi analisado a vazão total máxima de mensagens por segundo que o software é capaz de tratar, visando obter a capacidade máxima de transporte.

Em cada etapa, o teste foi repetido com tamanhos de mensagem diferentes, a fim de investigar os diversos comportamentos da implementação em resposta à variação na quantidade de bytes em cada mensagem. Os testes também foram duplicados, sendo eles realizados com variações nas configurações da rede, assim como no número de consumidores ao mesmo tempo. Esse teste tem por objetivo de visualizar a interferência externa e interna durante a execução.

Para a simulação do envio de mensagens por vários dispositivos IIoT, foi desenvolvido outro código em python que permite o usuário configurar a mensagem, suas características, número de dispositivos, assim como a frequência de envio. Os demais dados coletados vieram do próprio registro de execução implementado no demonstrador.

4.2 CENÁRIO COM CONSUMIDOR ÚNICO

Neste primeiro cenário de teste, foi realizado coleta de cada uma das métricas mencionadas em relação a um único consumidor inicializado. As subsecções a seguir foram divididas em estudos separados para a latência, vazão total e relação de perdas de mensagens.

4.2.1 Estudo sobre latência

No primeiro estudo deste cenário, o foco da análise recai sobre a aquisição das latências internas do software desenvolvido. Para este teste, foram realizadas quatro subdivisões visando investigar a influência do tamanho das mensagens no atraso do software. As mensagens de teste totalizaram 150 bytes, 5000 bytes, 20000 bytes e 40000 bytes, excluindo o tamanho do cabeçalho MQTT.

Como caracterização da rede, foi realizado teste utilizando a ferramenta Iperf3 versão 3.16, assim aferiu-se uma capacidade de transmissão média de 41,6 megabytes por segundo entre a máquina virtual e o hospedeiro. Com isso, sem considerar o

atraso de processamento da mensagem no *ThingsBoard*, latência de comunicação do hospedeiro com o *Docker* interno, atraso de fila e tempo de propagação, é possível calcular o tempo de atraso ideal, por fins de comparação, por meio da seguinte equação:

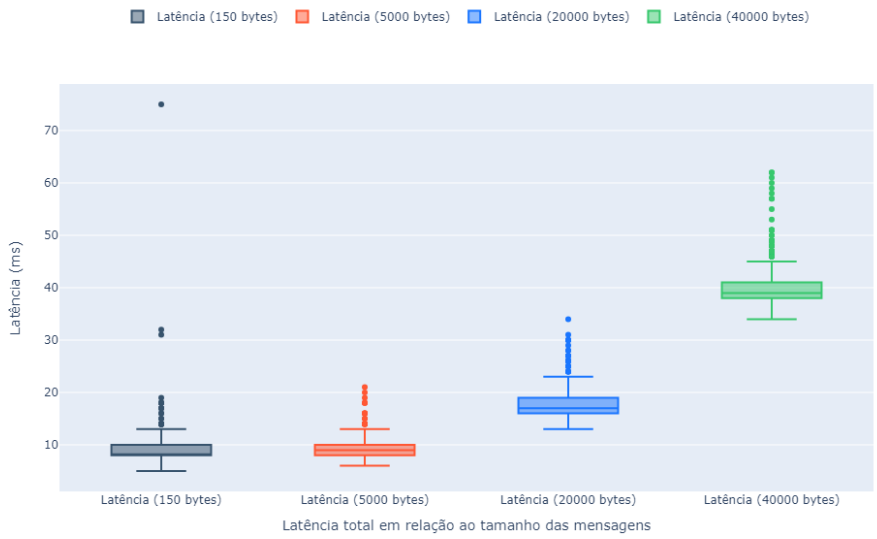
$$T = T_p + T_t + T_q + T_cpu \quad (1)$$

Contudo, como o método utilizado para realizar o envio das mensagens, simulando dispositivos IIoT, possui seu tráfego na mesma rede, a métrica de tempo ideal de resposta foi obtida com o cálculo baseado com metade da taxa de transmissão aferida. Com isso os campos dos testes abaixam apresentam um tempo de atraso ideal, calculado com uma taxa de transmissão de 20,8 megabytes por segundo. O tempo de retorno do ACK também foi desconsiderado uma vez que comparado ao tamanho das mensagens apresenta um tempo negligenciável.

Vale ressaltar que, durante a execução dos experimentos, as métricas de configurações da rede em questão foram alteradas conforme o passar do tempo. Essas mudanças nas características da rede têm como objetivo aprofundar o estudo do comportamento do demonstrador em condições adversas do ambiente em que se encontra.

Na Figura 22, é possível observar a latência total do sistema, desde o momento em que a mensagem é recebida no demonstrador, até o momento em que sua entrega é verificada pelo *ThingsBoard*. A cada 1000 mensagens enviadas, o tamanho da mensagem é modificado, sendo os tamanhos em bytes 150, 5000, 20000 e 40000, correspondentes às quatro divisões presentes no eixo x da figura.

Figura 22 – Resultado teste de latência



Fonte: Elaborado pelo Autor (2024)

Durante a entrega da primeira mensagem, pode se notar um pico de latência

em comparação com as demais. Isso se deve ao fato de que, ao enviar a primeira mensagem, o consumidor precisa conectar-se ao servidor, enquanto nas mensagens subsequentes ele já está conectado. No entanto, o atraso na entrega da primeira mensagem acarreta um atraso nas mensagens seguintes. É importante destacar que durante toda a execução do teste, não houve crescimento na fila do consumidor nem na fila responsável pelo redirecionamento das mensagens entre os processos do código.

Nesse contexto, a latência total do sistema pode ser composta pela latência das filas, responsáveis pelo encaminhamento das mensagens aos consumidores, somada à latência da confirmação de chegada da mensagem e o tempo de execução do código. Na Tabela 1, pode-se observar a média dessas latências e como elas variam conforme o tamanho de cada mensagem.

Tabela 1 – Dados de latências em ms, correlacionado ao tamanho das mensagens

Tamanho /Latência	Total	Fila	Desvio padrão total	ACK	ACK Ideal
150 bytes	8,984	4,018	3,00	3,739	0,007
5000 bytes	9,118	3,918	1,701	4,037	0,240
20000 bytes	17,465	4,273	2,448	11,851	0,961
40000 bytes	40,075	3,234	2,938	33,581	1,923

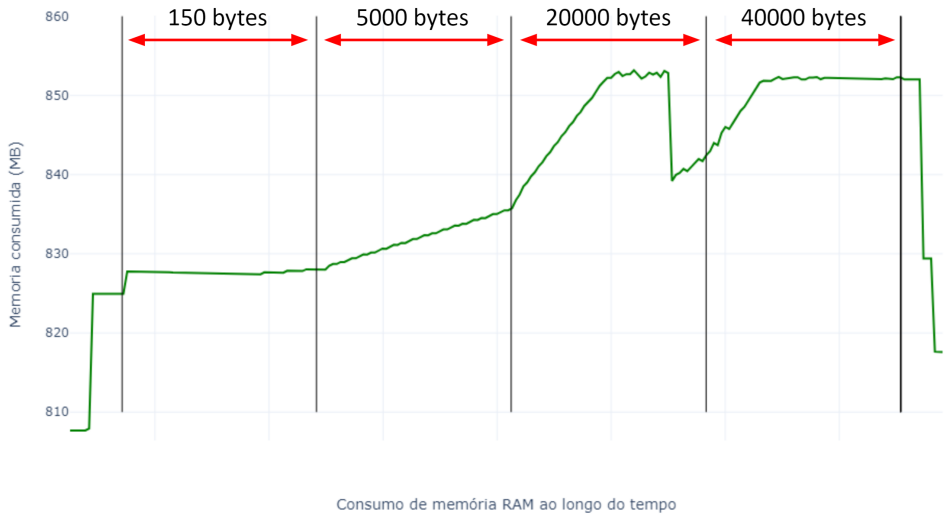
Fonte: Elaborado pelo Autor (2024)

Ao analisar as latências, para mensagens acima de 20 KB, é evidente que o atraso total do sistema é predominantemente causado pelo atraso na confirmação da chegada da mensagem. Uma vez que, a latência de redirecionamento das mensagens para outros processos, ou seja, a latência de fila, permanece relativamente constante em relação ao tamanho da mensagem, não sendo a principal influência no atraso total do sistema.

A comparação com o tempo de resposta ideal pelo ACK é feita de maneira simplória, pois certos atrasos são ignorados por motivos de simplificação, como mencionado no início da seção. Desta maneira, é importante ressaltar que essa simplificação não captura completamente a complexidade e os desafios reais enfrentados no contexto da comunicação da rede.

Outra validação relevante refere-se aos recursos computacionais consumidos durante a realização dos testes. Dentre as métricas analisadas, na Figura 23, é possível visualizar o consumo de memória RAM ao longo da execução do teste. Notavelmente, durante a fase em que mensagens de 20000 bytes são enviadas, ocorre uma queda no consumo de memória. Essa redução é atribuída ao gerenciamento de coleta de lixo.

Figura 23 – Resultado consumo de RAM



Fonte: Elaborado pelo Autor (2024)

Nota-se que, ao término do teste, ocorreu uma redução no consumo total de memória, decorrente da destruição do cliente pelo gerenciador ao atingir um limite de inatividade. Outra métrica coletada foi o uso de processamento pelo sistema. Conforme demonstrado na Tabela 2, observa-se um aumento no consumo de processamento em relação ao tamanho de cada mensagem.

Tabela 2 – Dados de uso de CPU em relação ao tamanho das mensagens

Tamanho das mensagens	Utilização média dos recursos da CPU (%)
150 bytes	11,117
5000 bytes	10,771
20000 bytes	17,043
40000 bytes	35,714

Fonte: Elaborado pelo Autor (2024)

Nesta etapa da validação, observa-se que o uso do processador não ultrapassa 35%. Vale ressaltar que nem todas as *Threads* do processador, necessariamente utilizam essa porcentagem a todo momento, uma vez que os dados apresentados na tabela são apenas uma média total das informações coletadas, as quais se referem às 4 *Threads* utilizadas pelo sistema.

4.2.2 Estudo de vazão total

A partir das taxas de latência total foram calculadas as vazões máximas de transferência para cada mensagem, os dados podem ser encontrados na Tabela 3. É

possível verificar que há uma diferença na taxa de transmissão conforme o tamanho de cada mensagem enviada em cada teste.

Tabela 3 – Vazão total do software em kilobytes por segundo

Tamanho das mensagens	Vazão em kilobytes/s
150 bytes	5,983
5000 bytes	548,365
20000 bytes	1145,147
40000 bytes	998,128

Fonte: Elaborado pelo Autor (2024)

Em um cenário onde o atraso proveniente de processamento é zerada, se esperaria que as taxas de transmissão total, não tenham correlação ao tamanho da mensagem. Contudo, no cenário criado para os testes, o processamento de cada mensagem necessita de tempo, ou seja, mesmo que a latência de transmissão seja proporcional ao tamanho da mensagem, o tempo necessário para o processamento, seja ele pelo trabalho proposto, pelo banco de dados e/ou pela plataforma *ThingsBoard* não é proporcional, o que acaba gerando uma grande diferença nas taxas de envio.

Embora com os dados atuais, um teste visando a aquisição da vazão total do sistema possa parecer redundante, o mesmo é proposto para a obtenção do motivo pelo qual é gerado o gargalo na comunicação. Nesse sentido, foi realizado um teste no qual a frequência de envio das mensagens foi aumentada gradativamente.

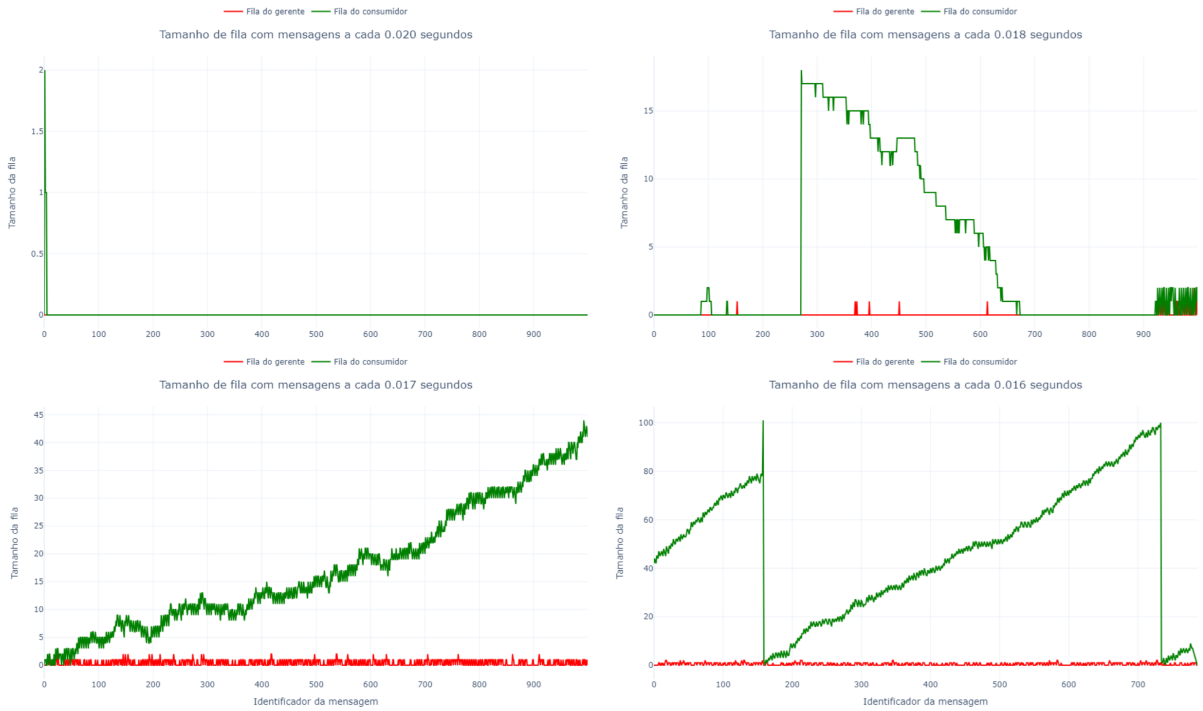
Para a realização do teste, foi adotado o tamanho de mensagem de 20 mil bytes, uma vez que esta apresentou a maior vazão verificada. Com base no tempo de latência total obtido no primeiro cenário de testes, foi definido o parâmetro de mensagens por segundo para o cenário atual.

A Figura 24 apresenta o comportamento da fila do consumidor e da fila do gerenciador de consumidores ao longo do tempo. Os diferentes gráficos presentes na imagem demonstram a evolução do tamanho da fila dos dois processos com base na alteração da frequência de envio.

No primeiro gráfico, ao adotar uma taxa de envio de 20 milissegundos, é observado um pico inicial de latência, originado pelo tempo requerido para um consumidor estabelecer conexão com o servidor. Durante o teste com intervalo de 18 milissegundos subsequentes, destaca-se um pico adicional de atraso, resultante da necessidade de retransmissão de uma mensagem utilizando o QoS 1. Essa ocorrência acarreta atrasos subsequentes para as mensagens subsequentes.

Durante o teste realizado com 17 milissegundos, é possível identificar que, a fila do consumidor apresenta um crescimento considerável, uma vez que ultrapassa a capacidade de envio do consumidor. No gráfico com o intervalo de envio de 16

Figura 24 – Tamanho de fila conforme mudança da frequência



Fonte: Elaborado pelo Autor (2024)

milissegundos, ao ultrapassar o limite do software, se nota um comportamento de gráfico de serra, visto que o consumidor possui em sua implementação um limite de fila de 100 mensagens.

Uma vez ultrapassando o limite de fila definido, o consumidor salva as mensagens no armazenamento local a fim de obter o mínimo possível de perdas de informações e minimizar o consumo de memória RAM. Contudo, isso não pode ser considerado na obtenção da vazão total, uma vez que as mensagens não chegam no banco de dados final.

4.2.3 Alteração da Latência

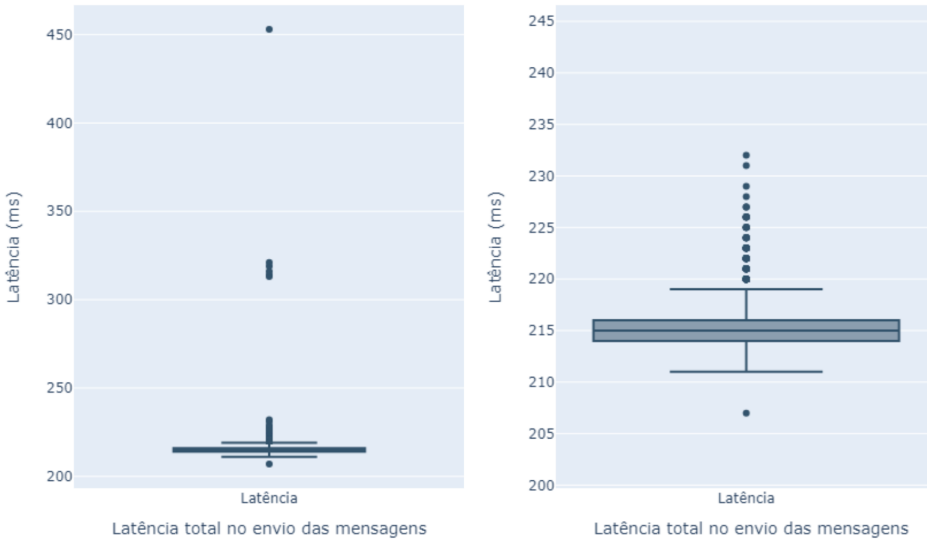
Para verificar a validar as métricas dos estudos anteriores, com relação a diferentes ambientes de rede, foi feito o uso da ferramenta TC/NETEM para efetuar alterações nas configurações do teste. Neste sentido, uma das métricas alteradas foi a adição de um atraso em todos os pacotes envolvidos entre a comunicação da máquina virtual com o *ThingsBoard*.

Como padronização, assim como adotado no teste de vazão total o tamanho de mensagens adotado para a realização do teste foi de 20 mil bytes. Com a adição de um atraso de 100 milissegundos, é esperado uma redução total na vazão de comunicação, assim como um atraso maior para o recebimento da confirmação de mensagem.

Na Figura 25, é possível observar as latências registradas durante o teste com a adição de atraso, uma ampliada e outra não. Como a adição de latência afeta

ambos os lados da comunicação, é esperado que tanto o envio da mensagem quanto o recebimento da confirmação sejam impactados.

Figura 25 – Resultado do teste com adição de latência



Fonte: Elaborado pelo Autor (2024)

Ao ampliar o gráfico, é possível notar que o comportamento das latências apresenta um padrão semelhante ao observado na seção do teste sem a adição de atraso. No entanto, observa-se um deslocamento total de 201,4 milissegundos a mais em relação ao teste anterior. Com isso, pode-se inferir que 100 ms afetam o envio da mensagem e outros 100 ms impactam o recebimento do ACK.

Como verificado na seção anterior, a taxa de vazão máxima, com um consumidor único, é limitada pelo tempo de atraso no recebimento da confirmação da entrega. Neste sentido, é possível calcular a vazão máxima por meio da média do atraso encontrado. Assim, foi evidenciado uma vazão total de 92,850 kilobytes por segundo.

4.2.4 Alteração da perda de pacotes

Em um ambiente de rede real, comumente é possível encontrar perda de pacotes nas comunicações realizadas (Guyon, 2019). Para analisar a capacidade de entregar todas as mensagens que passam pelo sistema foi implementado um teste com injeção de 10% de perda de pacotes.

Neste teste, assim como no teste de adição de latência, mensagens com o tamanho de 20 mil bytes foram enviadas. A frequência de envio foi mantida em prol de analisar um possível crescimento da fila dos consumidores com o esperado aumento de latência em pacotes perdidos na rede.

Na Tabela 8, são apresentados os resultados do teste realizado. Observa-se

um aumento de quatro vezes na média do tempo de atraso total. No entanto, ao analisar a mediana, que foi de 16,3 milissegundos, pode-se inferir que a média foi influenciada pela alta variação dos atrasos no envio, o que também é refletido no alto desvio padrão.

Tabela 4 – Dados do teste com injeção de perda de pacotes

Atraso Total	Desvio Padrão	Taxa de reenvio	ACKs não recebidos
95,633 ms	268,700 ms	9,584%	0,850%

Fonte: Elaborado pelo Autor (2024)

A presença de grandes variações nos resultados apresentados pode ser atribuída à realização do reenvio de mensagens por meio do QoS 1. Quando há perda de pacotes, uma determinada porcentagem do total de mensagens precisa ser reenviada, pois não foi possível sua chegada ao *Broker*, ou o reconhecimento dessas mensagens não pôde ser confirmado.

Ao contrário dos 10% de reenvios esperados, apenas 9,584% das mensagens precisaram ser reenviadas. Esse resultado pode ser atribuído à randomização na escolha dos pacotes afetados pela ferramenta TC/NETEM. Além disso, a ferramenta possui margens de erro, e o sistema operacional envia outros pacotes por diferentes processos, que também podem ser impactados pela perda. Essa variação compromete a isolação da perda apenas aos dados de teste, influenciando o resultado observado.

Contudo, durante os testes, algumas mensagens não tiveram seus ACKs reconhecidos, mesmo com a implementação do QoS 1. A porcentagem de mensagens que não foram entregues com sucesso foi de 0,85%. No entanto, como a implementação do demonstrador visa minimizar ao máximo a perda de mensagens, foi ativada a funcionalidade de salvamento local. Dessa forma, as mensagens não confirmadas foram armazenadas localmente e reenviadas posteriormente.

Com a obtenção do atraso total médio para a chegada da confirmação de entrega de cada mensagem, é possível obter a razão que indica a interferência da perda de pacotes na vazão total do sistema. Uma vez que, o limitante da vazão total é o atraso de entrega, se verifica uma diferença de 5,475 vezes na vazão total.

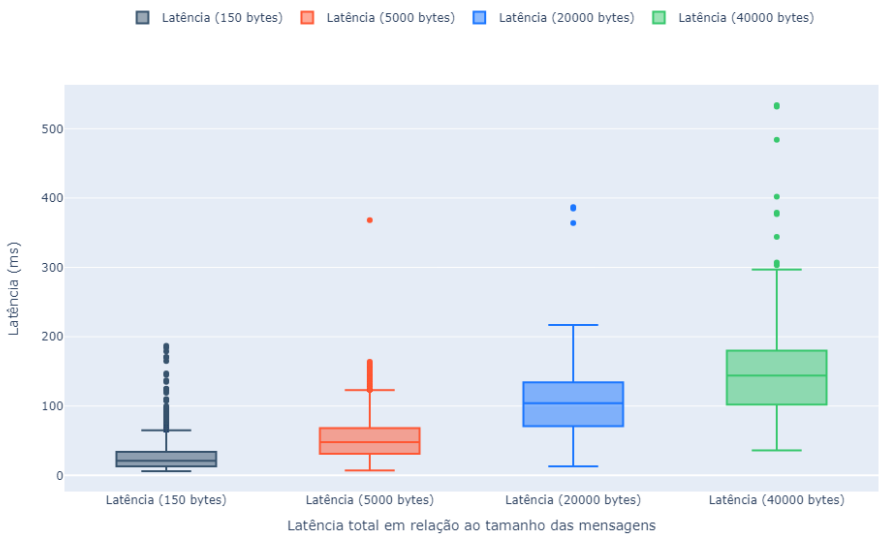
4.3 CENÁRIO COM CONSUMIDORES MÚLTIPLOS

O procedimento adotado para a realização do teste com múltiplos consumidores foi baseado no cenário com consumidor único. Contudo, neste teste foram adotados um total de 30 consumidores ativos, sendo deste 29 que participam no envio das mensagens, e um responsável pela checagem de conexão com o banco de dados.

4.3.1 Estudo sobre latência

Na Figura 26, se faz presente os atrasos retirados de uma bateria de testes. Nesse cenário, ao todo, foram enviadas 116 mil mensagens, 4 mil para cada um dos consumidores ativos, e registrado as latências de cada uma delas. Assim como no cenário de consumidor único, o tamanho das mensagens é variado conforme o passar dos testes.

Figura 26 – Resultado teste de latência



Fonte: Elaborado pelo Autor (2024)

Ao realizar a comparação com os atrasos do cenário de teste com consumidor único, é clara a diferença entre a latência total, que engloba a chegada da mensagem ao *Broker* local até o recebimento da confirmação de entrega pelo ThingsBoard. Na Tabela 5 pode-se encontrar detalhes específicos sobre a influência de cada um dos atrasos resultantes na latência total.

Tabela 5 – Dados de latências em ms, correlacionado ao tamanho das mensagens

Tamanho /Latência	Total	Fila	Desvio padrão total	ACK	ACK Ideal
150 bytes	25,068	7,057	9,450	15,144	0,007
5000 bytes	52,002	11,668	24,195	26,766	0,240
20000 bytes	102,722	22,725	59,090	39,823	0,961
40000 bytes	142,136	51,603	67,887	48,271	1,923

Fonte: Elaborado pelo Autor (2024)

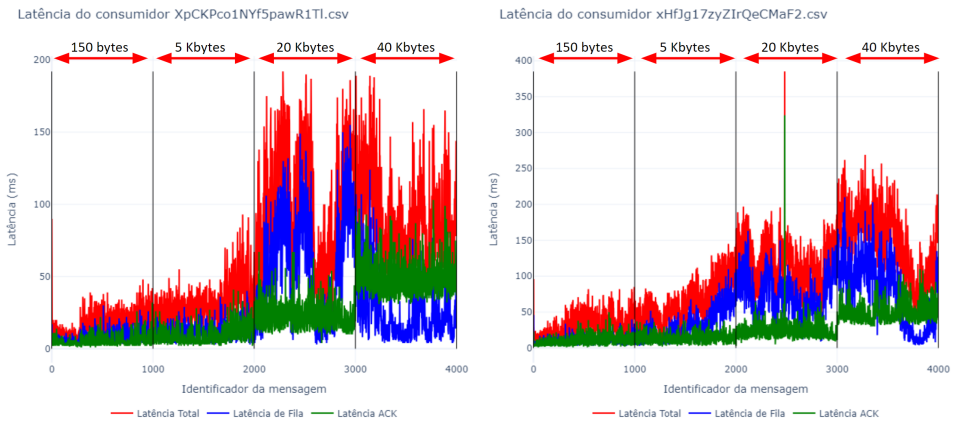
Assim como no cenário do consumidor único, a demora da confirmação de

entrega de mensagem, aumenta conforme o tamanho da mensagem. Contudo, a maior diferença está caracterizada pelo aumento significativo do atraso da fila, responsável pelo redirecionamento da mensagem para o processo dos consumidores. Entretanto, um aumento se é esperado pelo fato do gerenciador possuir mais do que um consumidor para manejar.

Neste cenário é possível notar que uma das mensagens com o tamanho de 40 KB possuem um atraso quatro vezes mais que a mediana. Isso é caracterizado pela perda da mensagem ou do reconhecimento da mesma durante o envio. Neste caso o uso da implementação de QoS 1 foi necessário para realizar o novamente o envio da mesma mensagem.

Com base nos resultados das variações do desvio padrão, é possível inferir que existe uma considerável diferenciação entre o envio das mensagens para diferentes consumidores. Tal disparidade é evidenciada na Figura 27, na qual se observa que, ao longo do envio das mensagens, embora o tratamento para ambos os consumidores seja uniforme, os tempos de latência de cada mensagem divergem significativamente.

Figura 27 – Resultado teste de latência



Fonte: Elaborado pelo Autor (2024)

Essa disparidade nos tempos de latência pode ser atribuída a uma série de fatores, incluindo variações na carga de trabalho dos consumidores, e a possível influência das flutuações de recursos do sistema. Além disso, a natureza dinâmica do ambiente de comunicação pode contribuir para essas variações, uma vez que as condições de rede e carga de trabalho dos consumidores podem mudar ao longo do tempo.

Com o intuito de isolar a variável dos recursos do sistema, a Tabela 6 apresenta a carga total imposta à CPU durante a execução do experimento. Observa-se que, embora o uso de processamento tenha aumentado durante o envio de mensagens com tamanho inferior a 40 KB, a porcentagem de utilização do processador para mensagens

exatamente de 40 KB permaneceu inalterada em comparação com o teste realizado com um único consumidor.

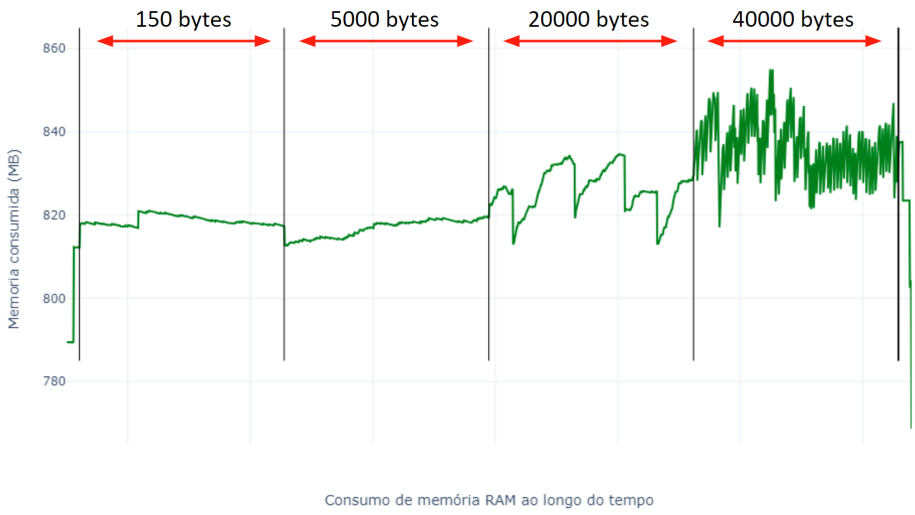
Tabela 6 – Dados de uso de CPU em relação ao tamanho das mensagens

Tamanho das mensagens	Utilização média dos recursos da CPU (%)
150 bytes	27,279
5000 bytes	26,544
20000 bytes	31,694
40000 bytes	35,502

Fonte: Elaborado pelo Autor (2024)

A Figura 28 ilustra a relação entre o consumo de memória RAM e a execução do teste proposto. Observa-se que o comportamento permanece consistente ao longo do tempo em relação ao teste com consumidor único, no entanto, com o número de consumidores elevado, a frequência da coleta de lixo pelo sistema também é ampliada. Essa tendência é especialmente evidente durante o envio de mensagens de maior tamanho, as quais ocupam mais espaço na memória.

Figura 28 – Consumo de RAM durante teste de latência



Fonte: Elaborado pelo Autor (2024)

4.3.2 Estudo sobre vazão total

A partir dos dados de latência total encontrados no cenário anterior é possível calcular a taxa de transferência de dados e com isso a vazão total permitida pelo software. Contudo, a fim verificar o gargalo do sistema, nesta seção será analisada o comportamento de fila conforme a vazão em cada um dos cenários anteriores.

Utilizando a mesma análise da taxa total de latência para calcular a vazão total do demonstrador com a aplicação de 29 clientes simultâneos se tem os resultados na Tabela 7. Neste caso pode se notar que a vazão máxima é dada pelo teste com mensagem de 40 kilobytes.

Tabela 7 – Vazão total do software em kilobytes por segundo

Tamanho das mensagens	Vazão em kilobytes/s
150 bytes	173,528
5000 bytes	2788,354
20000 bytes	5646,307
40000 bytes	8161,197

Fonte: Elaborado pelo Autor (2024)

A vazão de 40 mil bytes, identificada como a mais alta entre os testes, sugere que a principal limitação não reside mais no tempo de atraso do envio e da confirmação de recebimento das mensagens. Para investigar essa hipótese, foi concebido um teste que amplia a frequência na qual 29 consumidores enviam mensagens de 40 mil bytes, e as métricas do teste foram registradas.

Na Figura 29, são apresentados os dados da média das filas dos consumidores e do gerenciador de consumidores conforme a variação da frequência de envio de mensagens. Observa-se que, com uma frequência de envio de uma mensagem a cada 0,7 segundos por consumidor, tanto a fila do consumidor quanto a fila do gerente não apresentam crescimento ao longo do tempo.

Figura 29 – Tamanho de fila conforme alteração na frequência de envio

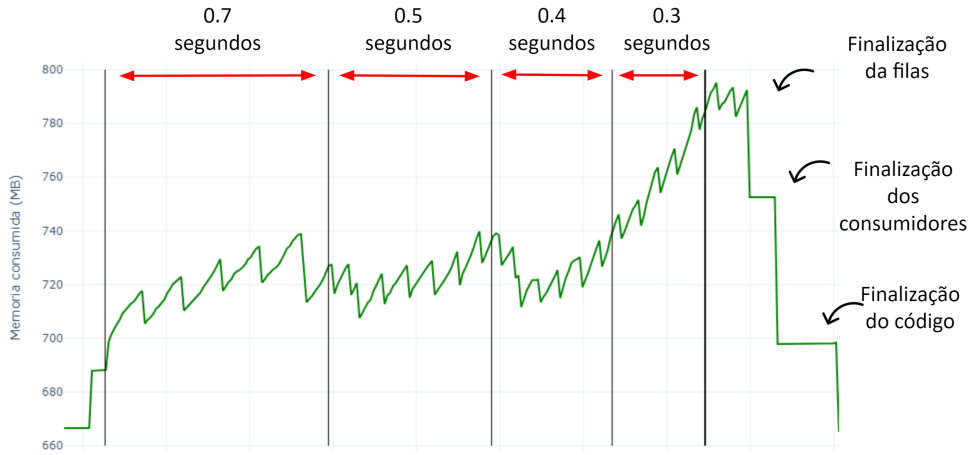
Fonte: Elaborado pelo Autor (2024)

Ao incrementar a taxa de mensagens para 0,5 e para 0,4 segundos, pode se notar frequentes alterações no tamanho de fila do gerenciador, enquanto a fila do consumidor se mantém com tamanho zerado. Isto é indicativo da proximidade do limite da vazão do demonstrador.

O último aumento na taxa de mensagens evidencia o limite total de vazão do software, já que a fila do gerenciador apresenta um crescimento não controlado. Assim, a frequência máxima de envio foi alcançada devido às limitações do gerenciador e não do consumidor, conforme demonstrado nos testes de vazão com um único consumidor.

Durante a execução destes testes, a métrica relacionada ao consumo de memória RAM evidência uma elevação em seu consumo a medida que a fila do gerenciador aumentava de tamanho. Na Figura 30, é possível notar que com o aumento da frequência de envio das mensagens, ao ultrapassar o limite de vazão o consumo de memória aumenta, fato dado pelo aumento da fila do gerenciador.

Figura 30 – Consumo de RAM durante teste de vazão.



Consumo de memória RAM conforme variação na frequência de envio

Fonte: Elaborado pelo Autor (2024)

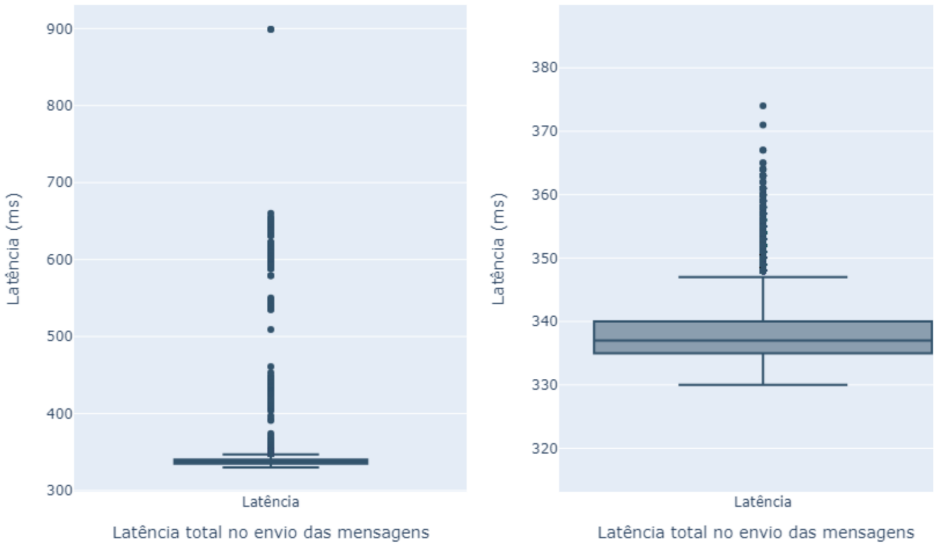
Na apresentação do consumo de memória RAM é possível delinear os espaçamentos no qual ocorreram a finalização das filas, assim como as finalizações dos consumidores que entraram em um período de inatividade. Por fim também é possível verificar a finalização do código do demonstrador, a qual cada uma dessas etapas influencia o comportamento no consumo de memória.

4.3.3 Alteração da Latência

Assim como no cenário do consumidor único, foi realizada a alteração de latência para visualizar o comportamento de múltiplos consumidores em funcionamento com a mudança desta grandeza. A implicação da escolha do atraso foi de maneira a padronizar com os testes do consumidor único, assim foi adicionado 100 milissegundos de atraso nas trocas de mensagens.

Neste sentido, a escolha do tamanho de mensagens foi de 40 mil bytes, tamanho o qual proporcionou a maior vazão em megabytes por segundo. Na Figura 31 é possível perceber que o envio de mensagens possui um comportamento parecido com o teste de latência sem alterações. Contudo, o mesmo é deslocado em 200 milissegundos.

Figura 31 – Resultado do teste com adição de latência



Fonte: Elaborado pelo Autor (2024)

Diferentemente do cenário com consumidor único, a taxa de vazão total não pode ser apenas calculada conforme o atraso médio. No cenário com múltiplos consumidores, o limitante a vazão total foi a capacidade de processamento do demonstrador. Vale ressaltar que neste cenário foi observado filas vazias, tanto na fila consumidor quando na fila do gerenciador.

Neste sentido, se faz necessidade de estudos futuros para descobrir a capacidade de transmissão máxima em cenários com a variação de latência. Uma vez que não foi observado o limite máximo de transferência do demonstrador no presente teste.

4.3.4 Alteração da perda de pacotes

A alteração na perda de pacotes foi também implementada no cenário com múltiplos consumidores, de modo a analisar o comportamento do demonstrador neste cenário. Assim, foi implantado um teste com mensagens de tamanho de 40 mil bytes e uma taxa de perdas de pacotes de 10%.

Tabela 8 – Dados do teste com injeção de perda de pacotes

Atraso Total	Desvio Padrão	Taxa de reenvio	ACKs não recebidos
251,425 ms	439,583 ms	10,284 %	1,242%

Fonte: Elaborado pelo Autor (2024)

Com a introdução da injeção de perda de pacotes é possível notar um aumento

no desvio padrão total observado na realização do teste. Esse valor do desvio padrão foi afetado pela perda do pacote de confirmação de entrega, pelo envio da mensagem e aumento de fila do consumidor.

Assim como no cenário de alteração de latência, neste estudo com consumidores múltiplos, se faz necessário uma análise mais aprofundada. De modo que a nova análise ajuste a vazão total do sistema em um ambiente com perda de pacotes.

4.4 CENÁRIO SIMULAÇÃO

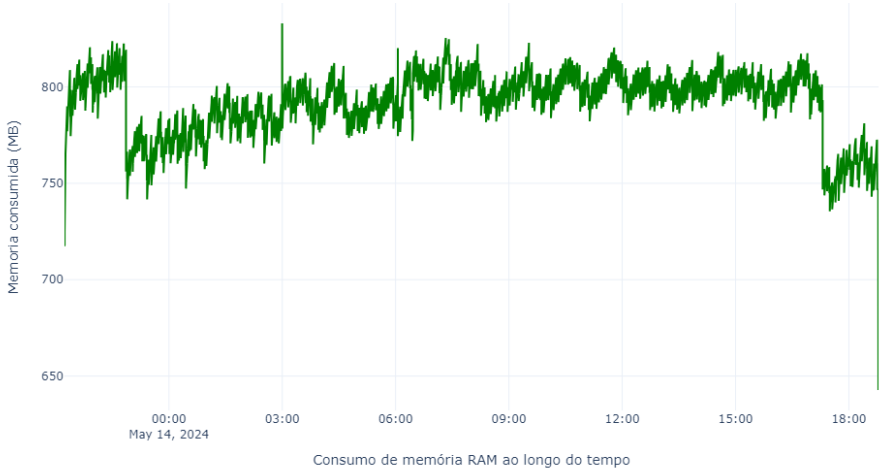
Para realizar testes que considerem perda de mensagens, assim como variação no número de consumidores e corrompimento de pacotes, um novo cenário foi criado. Este cenário busca realizar uma simulação de um possível ambiente real em uma rede com conectividade instável.

Neste contexto, utilizou-se a ferramenta TC/NETEM para forçar uma perda de pacotes na rede, configurada em 10%, e taxa de corrompimento de pacotes de 1%. Esta abordagem permitiu simular condições reais de rede instável, proporcionando uma análise mais precisa do comportamento do sistema sob essas circunstâncias. Assim, foi possível avaliar a robustez e a resiliência da comunicação frente a falhas na transmissão de dados.

Na aplicação desse caso de uso, foi feito o uso da quantidade limite de clientes suportados pela plataforma *ThingsBoard*. Portanto, clientes podem parar de realizar o envio de mensagens por um tempo randômico. Foi aplicado uma taxa de randomização do tamanho da mensagem enviada, sendo possível seu número de bytes variar de 150 a 40000.

O teste teve uma duração total de aproximadamente 21 horas, durante as quais um conjunto de mensagens foi enviado por 29 clientes. Na Figura 32, é possível observar o consumo de memória RAM ao longo da execução do teste, evidenciando como os recursos de memória foram utilizados em resposta ao tráfego gerado pelos clientes.

Figura 32 – Consumo de RAM durante teste de perdas



Fonte: Elaborado pelo Autor (2024)

O consumo ao longo do tempo de memória se apresentou estável durante a maioria da execução do teste em questão. Entretanto, pode-se notar duas quedas principais em determinados momentos. Em relação a isto, se faz necessário um estudo mais aprofundado em relação à dinâmica do uso de memória do sistema operacional com relação ao programa desenvolvido para obter conclusões mais precisas acerca deste assunto.

Neste caso, para obter métricas de taxa de reenvio, foi observado o parâmetro responsável pelo tempo em que o cliente guarda a mensagem para realizar o reenvio. Essa escolha foi feita para realizar a comparação entre a taxa de perdas de QoS 1 com a implementação do demonstrador.

Com uma implementação de QoS 0 no cliente MQTT, no cenário em questão, seria esperado uma taxa de perda de mensagens de aproximadamente 11%, sendo 10% proveniente da taxa de perda de mensagens e 1% da corrupção dos pacotes. Com a implementação do nível QoS 1, se espera que as perdas totais sejam reduzidas.

Assim, na Tabela 9 é possível ver que a taxa de reenvio foi superior a 11%. Uma hipótese para este acontecimento é, que, além da randomização dos pacotes afetados, caso o consumidor sofra um pico de desconexão no momento de recebimento de um ACK, o mesmo não é reconhecido.

Vale ressaltar, que as diferenças entre o cenário de perdas, ideal e real, é influenciado pelo ambiente de rede em questão. Com isso, a taxa de reenvio é só um indicativo de quantas mensagens necessitaram um novo envio, e não quantas mensagens foram perdidas.

A taxa de mensagens perdidas, mesmo após um possível reenvio por meio do

Tabela 9 – Dados do teste conexão instável

Total de mensagens	Taxa de reenvio	Taxa de ACKs não recebidos (pós QoS 1)
854525	11,613%	2,857%

Fonte: Elaborado pelo Autor (2024)

QoS 1, foi de 2,857%. Porém, esse número representa a percentagem de mensagens que não obtiveram confirmação de entrega, mesmo havendo a possibilidade de estarem presentes no banco de dados.

Esta taxa de perdas, é afetada em ambos os lados de comunicação, ou seja, o envio da mensagem pode estar comprometido, ou o recebimento do ACK. Desta forma, o demonstrador realizou o salvamento das mensagens sem confirmações de entrega no armazenamento local. Posteriormente, realizou o envio das mesmas, assim, a taxa final de perdas foi de 0% no ambiente com 10% de perda de pacotes e 1% de corrompimento.

4.5 CONSIDERAÇÕES PARCIAIS

Na presente seção, dedicada à análise do desempenho do demonstrador construído, foi empreendido diversos cenários para a aquisição de métricas relevantes. Contudo, vale ressaltar que a dinâmica apresentada nos resultados está vinculada ao contexto adotado.

As avaliações de métricas de consumo de memória RAM, uso de processamento, e até mesmo o comportamento da fila do gerenciador podem ser drasticamente alterados conforme o ambiente, uma vez que alterações de hardware afetam essas métricas diretamente. As diferenças de hardware também podem afetar até mesmo as latências de transmissão, uma vez que nelas estão implicitamente calculados os atrasos de processamento na comunicação.

As métricas relacionadas às latências e taxas de retransmissão também são sensíveis às características do ambiente de hardware em que foram observadas. Todavia, essas métricas são ainda mais suscetíveis às condições do ambiente de rede em que ocorre a comunicação.

Em diferentes contextos de redes, o comportamento de cada métrica apresentada nos resultados pode sofrer alterações que diferem dos resultados presentes. Isso deve a natureza das informações, cujo atraso está relacionado a veículos de transporte, reajuste de rotas de rede, controle de congestionamento e possíveis outras variáveis, visto a complexidade de um sistema de comunicação completo.

Dado isto, pode-se considerar válido os experimentos e resultados obtidos apenas para os cenários de teste delineados nesta seção. É crucial reconhecer que

as conclusões derivadas desses experimentos refletem especificamente as condições controladas e os parâmetros estabelecidos durante os testes. Portanto, qualquer generalização ou aplicação dos resultados para contextos além desses cenários requer uma análise cuidadosa e uma compreensão profunda das variáveis envolvidas.

4.6 LIMITAÇÕES

Nesta seção, são abordadas as restrições e limitações identificadas durante a análise dos resultados apresentados. Reconhecendo a complexidade do ambiente de rede e as escolhas realizadas na configuração dos parâmetros, esta seção fornece uma análise crítica das limitações do estudo, bem como das implicações e interpretações dos resultados alcançados.

É crucial destacar que o contexto em que a aplicação foi implementada apresenta uma relação intrínseca com as limitações encontradas. Nesse sentido, qualquer alteração no conjunto de hardware e na configuração da rede utilizados para a realização dos testes influencia diretamente as limitações observadas.

Uma das limitações do sistema é a capacidade máxima de clientes suportados, dado que a plataforma *ThingsBoard*, em sua versão gratuita, permite a conexão de no máximo 30 clientes. É importante destacar que o sistema em questão possui a capacidade de suportar um número maior de conexões. No entanto, ao exceder o limite imposto pela plataforma, um dos clientes conectados é desconectado automaticamente para permitir a conexão de um novo cliente.

Quando a capacidade de processamento não é suficiente para lidar com a quantidade de mensagens recebidas durante o teste de vazão máxima com múltiplos clientes, observa-se um aumento significativo no consumo de memória RAM. Esse crescimento decorrente do acúmulo na fila do gerenciador de consumidores. Esse comportamento não possui um modulo de controle na implementação do demonstrador.

Nesse sentido, a completude do requisito **RF-003** não foi bem sucedida. A implementação é capaz de suportar e isolar cada um dos consumidores. Contudo, o redirecionamento das mensagens recebidas para cada um dos consumidores, possui influência do tempo de fila causado pelo processamento do gerenciador de consumidores, como demonstrado no teste de vazão máxima.

5 CONCLUSÕES

Com o constante avanço das tecnologias inteligentes e a proliferação dos dispositivos IIoT nos ambientes industriais, a computação de borda emerge com o objetivo de aprimorar a eficiência e a segurança dos dispositivos conectados à rede local. No contexto industrial, otimizar o desempenho das comunicações e garantir um gerenciamento dos dados transmitidos são aspectos essenciais para a modernização, gerenciamento de dados, segurança e a competitividade do setor (Cao *et al.*, 2020).

Neste contexto, o objetivo deste trabalho consistiu no desenvolvimento de um software que desempenha o papel de gateway de comunicação. Esse gateway tem a capacidade de integrar dados oriundos de diversos dispositivos IIoT, os quais, empregam protocolos de comunicação como MQTT e OPC-UA. Além disso, o software garante a entrega desses dados a um banco de dados integrado, que por sua vez está conectado a um sistema de visualização, e em caso de falhas da comunicação a atual implementação salva os dados localmente para posteriormente serem enviados.

Na fase inicial do desenvolvimento do projeto, após as definições de requisitos, foi concebida uma arquitetura de estados voltada para o envio de mensagens MQTT a um banco de dados. Durante esta etapa, o principal foco foi o requisito funcional **RF-006**. Por meio da implementação de um mecanismo para armazenar mensagens que não obtiveram confirmação de recebimento, o software foi projetado para registrar tais mensagens, possibilitando tentativas de envio posterior.

Com a estruturação da arquitetura de um consumidor de mensagens concluída, a fase subsequente concentrou-se na elaboração de um mapa de processos capaz de gerenciar múltiplos consumidores simultaneamente. Essa etapa foi delineada com base no objetivo geral do trabalho, que consiste em proporcionar suporte para a conexão simultânea de diversos dispositivos IIoT. Essa abordagem possibilitou um ambiente escalável parar a gestão de diversas conexões.

Para analisar o funcionamento do sistema, foram elaborados múltiplos testes de validação, incluindo testes de latência, testes de vazão total de mensagens e avaliação da taxa de perdas. Nessa etapa, foram verificadas as limitações do sistema, como a latência adicionada, a taxa total de bytes por segundo que pôde ser enviada.

Um dos desafios enfrentados durante a execução do projeto envolveu o comportamento da implementação da plataforma Thingsboard. Essa plataforma impõe uma restrição na comunicação de um dispositivo específico com um determinado token de acesso, permitindo apenas uma única conexão simultânea com o mesmo nome de usuário e autenticação. Essa limitação exigiu que a metodologia de multiprocessamento utilizada no programa fosse ajustada em relação aos identificadores dos clientes. Como resultado, a vazão total de mensagens permitida pelo software foi reduzida.

Contudo, o trabalho proposto foi capaz de se comportar como uma aplicação cujo funcionamento serve de interface entre dispositivos IIoT e um banco de dados. Ao seguir a estruturação de mensagem proposta, o projeto em questão pôde integrar o recebimento do protocolo MQTT e OPC-UA, em seu modo Pubsub, por meio de uma única interface de entrada.

O demonstrador apresentou a característica de suportar o envio de múltiplos dispositivos IIoT em paralelo. A implementação também conseguiu gerar registros em tempo de execução, possibilitando o desenvolvimento de modelos estatísticos assim como a viabilidade das análises mencionadas.

Com os resultados dos testes, foi possível aferir a capacidade do demonstrado de garantir o envio da mensagem mesmo em um ambiente onde é possível identificar uma perda de pacotes superior a 11%. Assim, o objetivo geral, tanto como os objetivos específicos propostos foram atendidos.

Assim, em relação aos trabalhos futuros que poderão ser realizados a partir do estado atual do sistema projeto, considera-se a possibilidade de implementar uma metodologia que permita a cada cliente enviar e receber mensagens paralelamente, utilizando identificadores específicos para cada mensagem. Essa abordagem poderia resultar em um aumento significativo na taxa total de transferência de informações no sistema.

Outra possibilidade é a integração com outros protocolos de comunicação, como ModBus, ZigBee, AMQP, Bluetooth e RS-232. Desta maneira, é possível usar o trabalho em questão como base para a criação de um gateway IoT genérico capaz de lidar com diversos protocolos de comunicação por meio da mesma interface de comunicação.

REFERÊNCIAS

ARIF, S. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. **IEEE Internet of Things Journal**, v. 4, n. 6, p. 2148–2157, jan. 2017.

BANAFA, A. **Mastering internet of things**: building and implementing iot solutions. New York, NY: McGraw-Hill Education, 2019.

BANKS, R.; GUPTA, A.; HOSHINO, Y. **Mqtt essentials**: a lightweight iot protocol. New York, NY: O'Reilly Media, 2019.

BOSI, F. *et al*. Enabling smart manufacturing by empowering data integration with industrial iot support. *In:* **Proceedings** of the 2020 INTERNATIONAL CONFERENCE ON TECHNOLOGY AND ENTREPRENEURSHIP (ICTE). Bologna, 21-23 setembro. 2020. Disponível em: https://ieeexplore.ieee.org/document/9215538. Acesso em: 17 nov. 2023.

BRYNJOLFSSON, E.; MCAFEE, A. **The second machine age**: work, progress, and prosperity in a time of brilliant technologies. Nova Iorque: WW Norton & Company, 2014.

CALZADO, M. *et al*. Additive manufacturing technologies: an overview about 3d printing methods and future prospects. **Complexity**, v. 1, n. 1, p. 74–76, fev. 2019.

CAO, K. *et al*. An overview on edge computing research. **IEEE Access**, v. 8, n. 1, p. 85714–85728, jan./dez. 2020.

COELHO, P. M. N. **Rumo à indústria 4.0**. 2016. Dissertação (Mestrado em Engenharia Mecânica) — Faculdade de Ciências e Tecnologia de Coimbra, Coimbra, 2016.

CONJO, A. H. **Proposta de uma ferramenta web para assistência a modelagem de requisitos não funcionais de software**. Quixadá, 2022. Trabalho de Conclusão de Curso (Graduação em Engenharia de Software) — Campus Quixadá, Universidade Federal do Ceará, Quixadá, 2022.

FERREIRA, P. *et al*. Modelling and simulation in industry 4.0. **Springler**, v. 926, p. 57–72, fev. 2021.

FINANCE, A. Industry 4.0 challenges and solutions for the digital transformation and use of exponential technologies. **Finance**, v. 92, n. 1, p. 1–12, jan./dez. 2015.

GUYON, A. Augmented reality in industry: a review of potential applications. **Computers in Industry**, v. 108, n. 9, p. 102894, out. 2019.

HELLSTEN, T.; REPONEN, T. Opc: From data access to alarm and event management. **ISA transactions**, v. 41, n. 2, p. 221–234, nov. 2016.

HIVE-MQ. **MQTT-Specific Configurations Options**. 2024. Disponível em: https://docs.hivemq.com/hivemq/latest/user-guide/index.html.Acessoem:14Abril2024.

HUSEMANN, D.; ROß, M.; VOSSEBEIN, L. Opc specifications in a nutshell. **Control Engineering Practice**, v. 7, n. 12, p. 1497–1503, jan./dez. 1999.

JADALA, V. C. *et al*. Need of intenet of things, industrial iot, industry 4.0 and integration of cloud for industrial revolution. *In:* **Proceedings** of the 2021 INNOVATIONS IN POWER AND ADVANCED COMPUTING TECHNOLOGIES (I-PACT). Kuala, 08 de fevereiro. 2021. Disponível em: https://ieeexplore.ieee.org/document/9696696. Acesso em: 17 nov. 2023.

KANG, B. Security in the internet of things era: a comprehensive survey. **Journal of Computer and System Sciences**, v. 82, n. 8, p. 1423–1436, jan./dez. 2016.

KOURTIS, A. **MQTT vs. CoAP**: Battle of internet of things protocols. 2018. Disponível em: https://dzone.com/articles/mqtt-vs-coap-battle-of-iot-protocols. Acesso em: 17 nov. 2023.

LEE, J. Industrial big data analytics and cyber-physical systems for future maintenance & service innovation. **Procedia CIRP**, v. 38, n. 1, p. 3–7, jan./dez. 2015.

LIU, Z.; BELLOT, P. A configuration tool for mqtt based opc ua pubsub. *In:* **Proceedings** of the 2020 RIVF INTERNATIONAL CONFERENCE ON COMPUTING AND COMMUNICATION TECHNOLOGIES (RIVF). Ho Chi Minh City, 14 e 15 de outubro,. 2020. Disponível em: https://ieeexplore.ieee.org/document/9140792. Acesso em: 17 nov. 2023.

LUBUNTU. ***Lubuntu***. 2024. Disponível em: https://lubuntu.me/.Acessoem:02Maio2024.

LUCHIAN, R. A. *et al*. Iiot decentralized system monitoring for smart industry applications. *In:* **Proceedings** of the 2021 29th MEDITERRANEAN CONFERENCE ON CONTROL AND AUTOMATION (MED). Puglia, 15 de julho. 2021. Disponível em: https://ieeexplore.ieee.org/document/9480341. Acesso em: 17 nov. 2023.

MAHNKE, W.; LEITNER, S.-H.; DAM, M. **OPC Unified Architecture**. 1. ed. Berlin: Springer Science & Business Media, 2010.

MOSQUITTO. ***Eclipse Mosquitto***. 2024. Disponível em: https://mosquitto.org/ .Acessoem:01Maio2024.

MQTT. ***MQTT Specifications***. 2024. Disponível em: https://mqtt.org/mqtt-specification/ .Acessoem:14Abril2024.

OPC-FOUNDATION. **OPC Unified Architecture Specification**. 2021. Acesso em: 15 abril. 2024. Disponível em: https://opcfoundation.org/developer-tools/specifications-unified-architecture.

PAHO-MQTT. ***Paho Mqtt***. 2024. Disponível em: https://pypi.org/project/paho-mqtt/ .Acessoem:01Maio2024.

PARK, J.; KIM, H.; KIM, W. Dm-mqtt: an efficient mqtt based on sdn multicast for massive iot communications. **Sensors**, v. 18, n. 9, p. 3071, set. 2018.

PORTER, M. E.; HEPPELMANN, J. E. How smart, connected products are transforming competition. **Harvard Business Review**, v. 92, n. 1, p. 64–88, out. 2014.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH Editora, 2014.

PYTHON. ***Python***. 2024. Disponível em: https://www.python.org/.Acessoem: 02Maio2024.

QIU, T. *et al*. Edge computing in industrial internet of things: architecture, advances and challenges. **IEEE Communications Surveys Tutorials**, v. 22, n. 4, p. 2462–2488, jul. 2020.

RIOS, R. M. **Técnicas energeticamente eficientes para o posicionamento de aplicações em computação na borda**. Porto Alegre, 2022. Trabalho Acadêmico (Programa de Pós-Graduação em Ciência da Computação) — Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2022.

ROCHA, D. **Sistemas de supervisão e controlo de autómatos**: soluções baseadas em opc e iec 60870-5-104. 2013. Dissertação (Mestrado em Engenharia Mecânica) — Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto, 2013.

SANDEEP, K. K.; THAKUR, R. Mqtt protocol: working, advantages, disadvantages & applications. **International Journal of Computer Applications**, v. 181, n. 39, p. 16–21, jan./dez. 2018.

SCHWAB, K. **A quarta revolução industrial**. São Paulo: Edipro, 2016.

SHI, W.; CAO, J.; ZHANG, Q. Edge computing: vision and challenges. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 5, p. 637–646, jun. 2016.

SMITH, J. Autonomy in industry 4.0: enhancing operational efficiency. **Journal of Industrial Automation**, v. 15, n. 2, p. 45–62, jan./dez. 2020.

THINGSBOARD. ***ThingsBoard Open-Source IoT Platform***. 2024. Disponível em: https://thingsboard.io/.Acessoem:01Maio2024.

VSCODE. ***Microsoft Visual Studio Code***. 2024. Disponível em: https://code.visualstudio.com/.Acessoem:02Maio2024.

WANG, L. Big data analytics in smart manufacturing: a review. **Journal of Manufacturing Systems**, v. 48, n. 2, p. 1–13, jan./dez. 2018.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: a survey. **IEEE Transactions on Industrial Informatics**, v. 10, n. 4, p. 2233–2243, jan. 2014.