UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOSÉ VITOR MORENO

LUÍS HENRIQUE BELTRÃO SANTANA

ANÁLISE DE DESEMPENHO DE PROTOCOLOS PARA COMPUTAÇÃO EM NÉVOA

CURITIBA

2022

JOSÉ VITOR MORENO
LUÍS HENRIQUE BELTRÃO SANTANA

ANÁLISE DE DESEMPENHO DE PROTOCOLOS PARA COMPUTAÇÃO EM NÉVOA

PERFORMANCE ANALYSIS OF PROTOCOLS FOR FOG COMPUTING

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

Orientador: Profª. Drª. Ana Cristina Barreiras Kochem Vendramin

Coorientador: Prof. Dr. Daniel Fernando Pigatto

CURITIBA

2022

JOSÉ VITOR MORENO
LUÍS HENRIQUE BELTRÃO SANTANA

ANÁLISE DE DESEMPENHO DE PROTOCOLOS PARA COMPUTAÇÃO EM NÉVOA

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

Data de aprovação: 27/outubro/2022

Ana Cristina Barreiras Kochem Vendramin
Doutora
Universidade Tecnlógica Federal do Paraná

Mauro Sérgio Pereira Fonseca
Doutor
Universidade Tecnlógica Federal do Paraná

Paulo Roberto Bueno
Doutor
Universidade Tecnlógica Federal do Paraná

CURITIBA
2022

Dedicamos este trabalho às nossas famílias,
por toda a dedicação conosco.

AGRADECIMENTOS

Agradecemos às nossas famílias por tudo que fizeram por nós e pelo apoio que nos permitiu trilhar essa jornada durante a faculdade.

Aos nossos orientadores Ana Cristina Vendramin e Daniel Pigatto, pela orientação durante todo o processo de produção deste trabalho, tempo, correções e conhecimentos compartilhados conosco durante todo este período.

A todos os colegas que nos ajudaram, de diferentes formas, a superar inúmeros obstáculos que surgiram durante o curso.

Aos amigos que sempre estiveram presentes nestes anos.

É injusto não citar cada uma destas pessoas nominalmente, mas é mais injusto citar alguns e esquecer de outros. Foram tantas pessoas em tantos anos que a memória falha, mas saibam que são lembrados com carinho em nossas recordações.

RESUMO

Com o avanço do poder de processamento e a miniaturização dos processadores modernos, vê-se cada vez mais dispositivos portáteis se tornando populares. Esses dispositivos utilizam sistemas embarcados onde recursos como bateria, processamento e memória são escassos e de vital importância para que o usuário tenha uma boa experiência de uso. Além disso, os dispositivos móveis, os quais ficam situados nas bordas da rede, podem agir como grandes produtores de dados, sendo muitas vezes inviável tentar centralizar esses dados. Com o intuito de amenizar os problemas relacionados à escalabilidade de dados na borda da rede, a camada de computação em névoa é utilizada para intermediar os serviços de armazenamento e comunicação entre a computação em nuvem e os dispositivos finais, sendo capaz de fornecer dados em tempo real de forma descentralizada e escalável. Na computação em névoa pode existir uma entidade identificada como um servidor de mensagens, o qual é responsável por gerenciar e distribuir a informação através de um ou mais canais de comunicação, de modo a demandar menos recursos computacionais. Ao se fazer uso de um servidor de mensagens, também é necessário definir qual protocolo de comunicação será utilizado na transmissão de mensagens. Este trabalho busca analisar o desempenho de três protocolos de comunicação para a computação em névoa, como o MQTT (*Message Queuing Telemetry Transport*), AMQP (*Advanced Message Queuing Protocol*) e STOMP (*Simple Text Orientated Messaging Protocol*), empregando o RabbitMQ como servidor de mensagens. Com base nos resultados, pode-se observar que o MQTT obteve o melhor desempenho com relação ao consumo de energia e processamento, o AMQP consumiu menos memória e o STOMP apresentou um menor tempo para concluir a transmissão de cada mensagem.

Palavras-chave: computação em névoa; análise de desempenho; AMPQ; MQTT; STOMP.

ABSTRACT

With advancement in data processing and miniaturization in modern processors, mobile devices are becoming more popular. These devices use embedded systems with scarse resources such as battery, processing power and memory, which are important to provide a good user experience. Mobile devices, which are located at the edge of the network, could act as data producers and it is generally not feasible to centralize the data. In order to reduce problems related to data scalability at the edge of the network, the fog computing layer is used as a means of providing storage and communication services between cloud computing and mobile devices, providing real-time decentralized data in a scalable way. In fog computing there can be an entity called message broker, which is responsible for managing and distributing information through one or more communication channels in order to demand less computational resource. When using a message broker, it is also necessary to define which communication protocol will be used in the message transmission. This work aims to analyze some communication protocols for fog computing, such as MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol), and STOMP (SimpleText Oriented Messaging Protocol) working with RabbitMQ as message broker. Based on the results, it is possible to observe that MQTT presented the best performance regarding power consumption and processing, AMQP consumed less memory and STOMP presented a shorter time to complete the transmission of each message.

Keywords: fog computing; performance analysis; AMPQ; MQTT; STOMP.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Abreviaturas

ACK	abreviação do inglês *Acknowledgement*, que siginifica Confirmação

Siglas

AMQP	Protocolo avançado de enfileiramento de mensagens, do inglês *Advanced Message Queuing Protocol*
CoAP	Protocolo de aplicativo restrito, do inglês *Constrained Application Protocol*
CSV	Valores Separados por Vírgola, do inglês *Comma Separated Values*
DDS	Serviço de Dados Distribuídos, do inglês *Data Distribution Service*
ERP	Planejamento de Recursos Empresariais, do inglês *Enterprise Resource Planning*
HTTP	Protocolo de Transferência de Hipertexto, do inglês *Hypertext Transfer Protocol*
IaaS	Infraestrutura como serviço, do inglês *Infrastructure as a service*
IEC	Comissão Eletrotécnica Internacional, do inglês *International Electrotechnical Commission*
IoT	Internet das Coisas, do inglês *Internet of Things*
IP	Protocolo de Internet, do inglês *Internet Protocol*
ISO	Organização Internacional de Normalização, do inglês *International Organization for Standardization*
JSON	Notação de objeto JavaScript, do inglês *JavaScript Object Notation*
LwM2M	Máquina-para-Máquina leve, do inglês *Lightweight Machine-to-Machine*
M2M	Máquina-para-Máquina, do inglês *Machine-to- Machine*
MQTT	Enfileiramento de mensagens, transporte e telemetria, do inglês *Message Queuing Telemetry Transport*
MQTT-SN	Enfileiramento de mensagens, transporte e telemetria para sensores de network, do inglês *Message Queuing Telemetry Transport for Sensor Network*

MOM	Middleware Orientado a Mensagem
MVVM	Modelo-Visualização-ModeloVizualização, do inglês *Model-View-ViewModel*
NATO	Organização do Tratado do Atlântico Norte, do inglês *North Atlantic Treaty Organization*
OASIS	Organização para o Avanço dos Padrões de Informação Estruturada, do inglês *Organization for the Advancement of Structured Information Standards*
OPC UA	Arquitetura unificada OPC, do inglês *OPC Unified Architecture*
OSI	Interconexão de Sistemas Abertos, do inglês *Open System Interconnection*
PaaS	Plataforma como serviço, do inglês *Plataform as a service*
QoS	Qualidade de Serviço, do inglês *Quality of Service*
RFID	Identificação por Radiofreqüência, do inglês *Radio Frequency Identification*
SaaS	Software como serviço, do inglês *Software as a service*
SQL	Linguagem de Consulta Padrão, do inglês *Standard Query Language*
STOMP	Enfileiramento de mensagens orientado a texto simples, do inglês *Simple Text Orientated Messaging Protocol*
TCP	Protocolo de controle de transmissão, do inglês *Transmission Control Protocol*
UDP	Protocolo de controle de usuário, do inglês *User Control Protocol*
UTF-8	Formato de Transformação 8 do USC, do inglês *UCS Transformation Format 8*
WAMP	Servidor Wamp, do inglês *WampServer*
WSN	Serviço Web de Notificação, do inglês *Web Services Notification*
XMPP-IoT	Messalina Extensível e Protocolo de Presença para IoT, do inglês *Extensible Messalina and Presence Protocol for IoT*

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis estão se tornando cada vez mais comuns no dia a dia das pessoas. Muitas vezes esses dispositivos são capazes de realizar tarefas que anteriormente eram apenas executáveis por grandes computadores. Isso se deve à miniaturização dos processadores e ao aumento do poder computacional (SILVA, 2017; COUTINHO, 2014).

Esse poder de processamento, associado ao pequeno tamanho dos dispositivos, permite trabalhar com a troca de informações entre vários aparelhos. A produção e consumo de dados, feita por esses dispositivos, pode facilmente atingir a ordem dos milhares em um espaço muito curto de tempo (AZEVEDO, 2017; MAGNONI, 2014).

Com uma grande quantidade de dados para serem gerenciados e distribuídos, surgiu a necessidade de fazer transmissões de dados de forma mais organizada e escalável, em uma rede descentralizada formada por vários dispositivos. Porém, os dispositivos móveis possuem limitações com relação à disponibilidade de energia, o que também afeta a disponibilidade de memória e processamento (MAGNONI, 2014; SILVA, 2017).

Para entender como os dispositivos móveis se comportam em uma rede descentralizada e escalável, este trabalho estudou o uso dos protocolos STOMP, AMQP e MQTT em computação em névoa. Os protocolos foram analisados quanto a perda de mensagens, tempo total de ida e volta de mensagens, consumo de energia, consumo de memória e uso de processamento.

1.1 Motivação

Atualmente muitas informações são geradas de forma descentralizada e por um grande número de produtores. Isto cria a necessidade de conhecer o endereço na rede de cada produtor de dados, para que assim seja possível estabelecer comunicação com eles (MAGNONI, 2014).

Uma forma de resolver esse problema de geração descentralizada de dados é criar um canal de comunicação. Neste canal os produtores publicam suas mensagens e os consumidores, que têm interesse, recebem informações dele (MAGNONI, 2014).

Com o aumento no uso de dispositivos móveis, esses dados passam a ser produzidos e consumidos em sistemas onde os recursos são limitados. Portanto, é importante analisar o comportamento dos protocolos de comunicação em névoa com relação a essas limitações.

Este trabalho visa analisar o comportamento dos protocolos de comunicação MQTT (*Message Queuing Telemetry Transport*) (OASIS, 2014), AMQP (*Advanced Message Queuing Protocol*) (RABBITMQ, 2011) e STOMP (*Simple Text Orientated Messaging Protocol*) (STOMP, 2013) em dispositivos móveis com relação ao uso de recursos escassos como energia, processamento e memória, tempo de ida e volta de mensagens e taxa de perda de mensagens.

1.2 Objetivo Geral e Objetivos Específicos

O presente estudo tem como objetivo investigar experimentalmente o comportamento dos protocolos AMQP, MQTT e STOMP em dispositivos móveis executando o sistema operacional Android 12. Os objetivos específicos deste trabalho são:

- Analisar o desempenho dos protocolos AMQP, MQTT e STOMP quanto ao consumo de energia e à duração do experimento de energia em função do tempo entre mensagens;
- Analisar o desempenho dos protocolos AMQP, MQTT e STOMP quanto ao consumo de memória, uso do processador, perda de mensagens e tempo de ida e volta em função do tamanho das mensagens enviadas;
- Identificar quais dos protocolos (AMQP, MQTT e STOMP) são mais recomendados para cada uma das métricas análisadas.

1.3 Organização do documento

O presente documento está organizado em cinco capítulos. O Capítulo 2 apresenta o referencial teórico e os trabalhos relacionados. O Capítulo 3 detalha o projeto de software do sistema criado para realizar os experimentos de desempenho do dispositivo móvel. O Capítulo 4 apresenta a metodologia utilizada para os experimentos, os resultados obtidos e suas respectivas análises. O Capítulo 5 contém as conclusões.

2 REFERENCIAL TEÓRICO

Este capítulo contém o embasamento teórico necessário para o entendimento deste trabalho. As Seções 2.1, 2.2 e 2.3 apresentam conceitos de *Internet* das Coisas, Computação em Nuvem e Computação em Névoa, respectivamente. As Seções 2.4, 2.4.1 e 2.4.2 descrevem o modelo publicação/subscrição de eventos, o servidor de mensagens e os protocolos de comunicação analisados neste trabalho, respectivamente. A Seção 2.5 apresenta os trabalhos relacionados.

2.1 Internet das Coisas

Nos últimos anos houve um aumento significativo no uso de aplicações e sistemas computacionais que seguem o paradigma de *Internet* das Coisas (IoT, do inglês *Internet of Things*). Segundo Patel e Patel (2016) e Johnsen (2018), este termo foi cunhado pela primeira vez pelo britânico Kevin Ashton ao se referir ao uso de RFIDs (Identificadores por Frequência de Rádio, do inglês *Radio-Frequency IDentification*) no mundo.

Atualmente, o termo IoT refere-se a uma rede de objetos físicos que executam uma aplicação com capacidade de acesso à *Internet*. Porém, a busca por expandir esse paradigma trouxe uma série de desafios a serem resolvidos, que segundo Patel e Patel (2016) são: interconectividade, confiabilidade, heterogeneidade, escalabilidade, segurança, conectividade, entre outros.

Uma “coisa” em IoT se refere a qualquer objeto físico que possa ser controlado, que armazene dados ou que seja capaz de estabelecer comunicação com outros objetos (JOHNSEN, 2018). Foi devido ao crescente uso de dispositivos móveis mais poderosos, do surgimento da computação em nuvem e da diminuição do custo de sensores que a IoT tem se popularizado ao longo dos últimos anos (JOHNSEN, 2018).

A computação móvel tem como objetivo atender a demanda de se desenvolver soluções para dispositivos móveis, que normalmente utilizam redes sem fio para se conectar à *Internet*.

Atualmente, o uso de dispositivos móveis já está amplamente difundido na sociedade, inclusive alterando a forma como ela se comporta e se organiza. O uso de telefones móveis, por exemplo, aumentou exponencialmente nos últimos anos, principalmente entre as pessoas mais jovens (SHUIB; SHAMSHIRBAND; ISMAIL, 2015). Algumas das vantagens da computação móvel são: portabilidade, acesso rápido às informações na *Internet*, uso eficiente de tempo, comunicações flexíveis, aplicativos poderosos e acesso a recursos multimídia (WALLACE; CLARK; WHITE, 2012). Esses fatores citados justificam a atual demanda pela criação de soluções para os problemas desse tipo de computação.

Apesar do uso constante de aplicações e dispositivos relacionados à computação móvel, ainda existem alguns problemas que dificultam a implementação desse tipo de computação, que segundo Satyanarayanan (1996) são:

- Dispositivos móveis, geralmente, têm menor capacidade de processamento e de armazenamento de dados comparados a dispositivos fixos;
- A possibilidade do equipamento ser furtado ou danificado é maior;
- A conexão com a rede móvel não é constante, o que afeta a performance e a confiabilidade;
- Dispositivos móveis têm baterias com energia limitada, o que afeta o planejamento no uso de recursos do aparelho.

Vários tipos de soluções computacionais foram desenvolvidas até o momento, agregando novas funcionalidades para sistemas de IoT. Um tipo comum é o uso da computação em nuvem em sistemas que usam, ou disponibilizam, uma grande quantidade de recursos via *Internet* (AMANATULLAH *et al.*, 2013). Outra solução mais recente foi a criação da computação em névoa, que é uma camada intermediária entre a camada de computação em nuvem e os dispositivos que a utilizam, por exemplo, as “coisas” em sistemas IoT (SARKAR; CHATTERJEE; MISRA, 2015).

A computação em nuvem e a computação em névoa estão descritas nas seções 2.2 e 2.3, respectivamente.

2.2 Computação em Nuvem

A computação em nuvem se caracteriza por tipos de serviços disponibilizados via *Internet* para o uso em aplicações. Segundo Amazon (2018) este tipo de computação é utilizada quando não há interesse por parte dos criadores da aplicação em manter certos recursos implementados localmente, seja por limitações de projeto ou por praticidade. Desta forma, os serviços em nuvem são entregues sob demanda para cada aplicação que os utiliza.

Segundo NIST (2011) os serviços de computação em nuvem podem ser disponibilizados nas seguintes formas:

- Infraestrutura como serviço (IaaS, do inglês *Infrastructure as a service*): nesta modalidade, o consumidor pode utilizar do processamento, armazenamento, rede e outros recursos computacionais do serviço, onde é possível implementar e executar certos softwares, que podem incluir sistemas operacionais e aplicações variadas;
- Plataforma como serviço (PaaS, do inglês *Plataform as a service*): neste caso, o consumidor tem a possibilidade de implantar, em uma infraestrutura na nuvem, aplicações criadas com o uso de linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. Essas aplicações podem ser criadas pelo usuário ou adquiridas prontas. O usuário não consegue controlar a infraestrutura interna, quem o faz é o provedor;

- *Software* como serviço (SaaS, do inglês *Software as a service*): nesta modalidade, o consumidor deve ter a possibilidade de utilizar a aplicação do provedor, que é executada em uma infraestrutura em nuvem. O usuário não consegue controlar a infraestrutura interna do serviço, pois quem tem esse controle é o provedor do serviço.

2.3 Computação em Névoa

Computação em névoa é uma tradução do termo em Inglês “Fog (From cOre to edGe) Computing”, cunhado pela primeira vez pela Cisco em 2012 (SARKAR; CHATTERJEE; MISRA, 2015). Consortium (2017) define a computação em névoa como uma arquitetura horizontal, capaz de distribuir poder computacional, armazenamento e conexão de *Internet* para os usuários do serviço através da rede, que se estende desde a nuvem até a “coisa” em IoT.

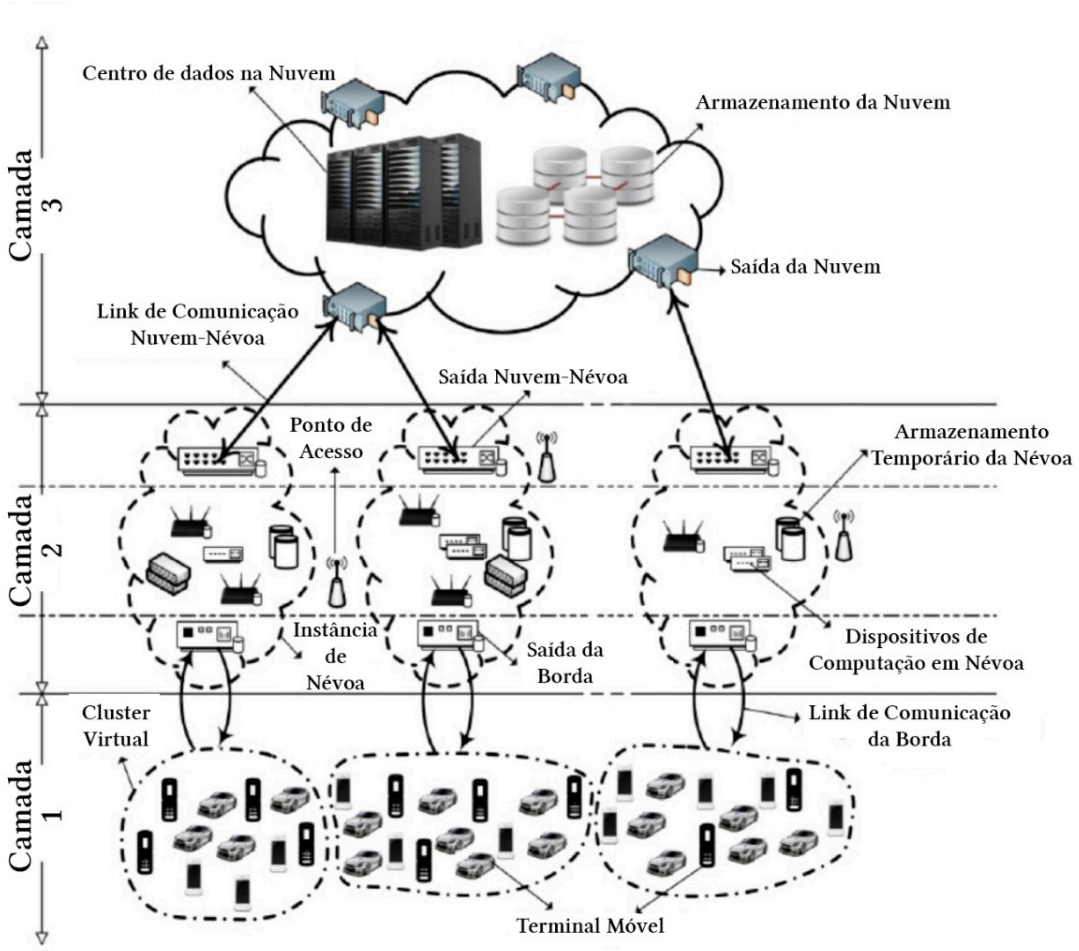
A computação em névoa permite estender recursos da computação em nuvem para mais perto da borda da rede, com a intenção de satisfazer alguns requisitos que o modelo centralizado da computação em nuvem não consegue. Portanto, um modelo em névoa permite fornecer dados, processamento, armazenamento e serviços de forma mais próxima ao usuário. O uso de computação em névoa reduz o tráfego e o processamento dos dados em uma infraestrutura em nuvem, tornando os dados utilizados mais significativos para as aplicações (COUTINHO; CARNEIRO; GREVE, 2016).

Sendo assim, a computação em névoa é uma extensão da computação em nuvem, onde a informação está mais próxima dos artefatos que agem no mundo físico. Além disso, o modelo em névoa permite que aplicações e serviços que não se adequam completamente em uma estrutura em nuvem sejam tratados de forma mais adequada. De acordo com CISCO (2015) os dois maiores objetivos da computação em névoa são a eficiência em redes IoT e, também, a capacidade de utilização de mecanismos que geram dados em tempo real para cenários heterogêneos.

Segundo CISCO (2015), os dispositivos na névoa, chamados de nós da névoa, podem ser implementados em qualquer lugar, desde que tenham conexão com a *Internet*. Qualquer aparelho com conectividade à *Internet*, poder de computação e armazenamento pode ser um nó de névoa (alguns exemplos são *switches*, roteadores e câmeras de vídeos).

A Figura 1 apresenta o modelo de Computação em Névoa. A camada 1 contém as “coisas”, dispositivos computacionais físicos, comuns em um sistema IoT. A camada 3 apresenta a computação em nuvem com seus servidores centralizados. Já a camada 2 se refere à computação em névoa, que é a camada intermediária entre a IoT e a nuvem.

Figura 1 – Exemplo de Computação em Névoa



Fonte: Adaptado de Sarkar, Chatterjee e Misra (2015).

2.4 Modelo publicação/subscrição

Sistemas distribuídos baseados em eventos possibilitam que objetos, em diferentes locais, registrem o seu interesse em receber notificações sobre eventos ocorridos em outro objeto (COULOURIS; DOLLIMORE; KINDBERG, 2011). Segundo O'Riordan (2020b) e O'Riordan (2020a), alguns exemplos destes sistemas são:

- Dispositivos utilizados por motoristas podem realizar subscrições por informações de trânsito e de rotas;
- Sistemas de despacho e ERP (Planejamento de Recursos Empresariais, do inglês *Enterprise Resource Planning*) podem realizar subscrições para receber atualizações de entregas concluídas;
- Sistemas de rastreamento e despacho podem obter atualizações de posição em tempo real quando necessário;

- Um participante pode se subscrever em um determinado bate-papo online com uma aplicação para receber mensagens enviadas a este bate-papo.

Sistemas baseados em eventos usam o modelo publicação/subscrição, que é um modelo considerado como um paradigma de comunicação de um-para-muitos (COULOURIS; DOLLIMORE; KINDBERG, 2011).

No modelo publicação/subscrição há dois tipos de clientes: os publicadores e os assinantes/consumidores, que serão referidos neste documento como produtores e consumidores, respectivamente. O produtor publica eventos estruturados no serviço de eventos e o consumidor faz subscrições neste serviço, indicando que tem interesse em determinados tipos de eventos (COULOURIS; DOLLIMORE; KINDBERG, 2011).

Ainda segundo Coulouris, Dollimore e Kindberg (2011), este modelo apresenta algumas características que podem ser interessantes, dependendo dos requerimentos da aplicação, tais como:

- Comunicação estabelecida de forma assíncrona;
- Comunicação entre sistemas heterogêneos;
- Garantia de entrega de notificações;
- Informações atualizadas nas duas pontas da comunicação;
- Possibilidade de ser implementado dentro de um sistema de comunicação em tempo real, com confiabilidade e ordenamento que satisfazem propriedades de um sistema distribuído síncrono.

Com base no modelo de subscrição (filtro) utilizado, os sistemas publicação/subscrição podem ser classificados em quatro categorias (COULOURIS; DOLLIMORE; KINDBERG, 2011):

- Baseada em canais: nessa categoria existem canais específicos onde os produtores publicam eventos, enquanto os consumidores se subscrevem nos canais que tem interesse em receber notificações de todos os eventos que chegarem neles;
- Baseada em tópicos: essa categoria é similar à categoria baseada em canais. A diferença é que nesta categoria a definição do tópico é mais explícita, onde no momento em que o evento é publicado também é indicado a qual tópico ele pertence. Os consumidores por sua vez indicam de quais tópicos tem interesse em receber notificação. Cada notificação é expressa em termos do número de campos, onde cada campo denota o tópico. Tópicos podem ser organizados de forma hierárquica;
- Baseada em conteúdo: considerado como uma generalização da categoria anterior. Neste tipo, a separação de eventos é feita a partir de composições de restrições sobre os valores dos atributos do evento;

- Baseada em tipo: o consumidor indica quais tipos de eventos são de seu interesse, o que permite a criação de um filtro com relação aos tipos e subtipos dos eventos publicados no serviço de eventos. Esse tipo de categoria está diretamente relacionada a abordagens baseadas em objetos, onde os objetos têm um tipo especificado.

Nos processos de comunicação entre produtores e consumidores, é comum o uso de servidores de mensagens (em inglês *Message Broker*). Segundo Kale (2014), um servidor de mensagens é um *Middleware* Orientado a Mensagem (MOM) que funciona como uma infraestrutura do tipo cliente/servidor que intermedia a comunicação entre aplicações. Os clientes desse tipo de serviço são definidos de duas formas: os publicadores são clientes que enviam as mensagens ao servidor de mensagens; e os consumidores são clientes que consomem as mensagens armazenadas no servidor de mensagens. Este tipo de infraestrutura está descrita com mais detalhes na próxima seção.

2.4.1 Servidor de Mensagens

Um servidor de mensagens permite que aplicações heterogêneas possam trocar dados, de qualquer tipo, de forma assíncrona, seguindo o paradigma publicação/subscrição. Através dele, produtores e consumidores não precisam estar disponíveis ao mesmo tempo para se comunicarem. Isto ocorre porque as mensagens enviadas por um produtor podem ser armazenadas até o momento em que o consumidor as possa receber (COULOURIS; DOLLIMORE; KINDBERG, 2011).

Segundo Kale (2014), o uso de um servidor de mensagens concede alguns benefícios, como:

- Cooperação transparente entre sistemas heterogêneos;
- Requisições com maior prioridade sendo tratadas antes das com menor prioridade;
- Controle de fluxo e *buffering* de mensagens por meio de filas;
- Mensagens persistentes, ou seja, é garantido que o cliente receba a mensagem pelo menos uma vez;
- Flexibilidade e confiabilidade, já que o uso de comunicação assíncrona permite um balanceamento de carga flexível e dinâmica;
- Escalabilidade e uso otimizado de recursos.

Cada servidor MOM pode ter seus próprios protocolos de comunicação, que estão detalhados a seguir.

2.4.2 Protocolos de Comunicação

Um protocolo de comunicação é um conjunto de regras que permite que dois ou mais dispositivos estabeleçam um sistema de comunicação confiável, possibilitando a troca de informações entre os dispositivos (IRONS-MCLEAN; SABELLA; YANNUZZI, 2019). Esse conjunto de regras define sintaxe, semântica, meios de sincronização e mecanismos de tratamento em estado de erro (IRONS-MCLEAN; SABELLA; YANNUZZI, 2019). Existem protocolos de comunicação que foram padronizados para uso livre da comunidade, enquanto outros são para uso privado (ARDC, 2020).

Existem muitos protocolos de comunicação voltados a sistemas IoT, porém, alguns são mais utilizados que os outros. Segundo Wytrebowicz, Cabaj e Krawiec (2021), os protocolos de comunicação para *Internet* das Coisas mais populares são:

- MQTT (*Message Queuing Telemetry Transport*) (MQTT.ORG, 2022);
- MQTT-SN (*MQTT for Sensor Network*) (SOUZA, 2018);
- CoAP (*Constrained Application Protocol*) (BORMANN, 2016);
- STOMP (*Simple (or Streaming) Text Orientated Messaging Protocol*) (STOMP, 2013);
- XMPP (*Extensible Messaging and Presence Protocol*) (XSF, 2022);
- OPC (*Open Platform Communications*) (FOUNDATION, 2022);
- WAMP (*Web Application Messaging Protocol*) (CROSSBAR.IO, 2022);
- AMQP (*Advanced Message Queuing Protocol*) (OASIS, 2022b);
- DDS (*Data Distribution Service*) (DDS, 2022);
- LwM2M (*Lightweight M2M*) (SPECWORKS, 2022);
- Weave (*OpenWeave*) (NEST, 2022).

O desempenho dos protocolos MQTT, AMQP e STOMP são analisados neste trabalho. Esses protocolos foram escolhidos pelas suas popularidades e pela existência de suporte nativo a eles no servidor de mensagens RabbitMQ (RABBITMQ, 2022a), o qual é utilizado no presente trabalho. Esses três protocolos são detalhados nas próximas subseções.

2.4.2.1 MQTT

De acordo com Cope (2018), o protocolo MQTT foi desenvolvido em 1999 por Andy Stanford-Clark e Arlen Nipper para transmissão de dados de telemetria de oleodutos para satélites. Atualmente, esse protocolo pode ser livremente utilizado e é padronizado pelo OASIS

(OASIS, 2022a), que é uma organização sem fins lucrativos que define padrões internacionais. Segundo Naik (2017), o MQTT é um protocolo que utiliza o modelo publicação/subscrição e o protocolo TCP (Protocolo de Controle de Transmissão, do inglês *Transmission Control Protocol*) na camada de transporte.

As versões mais atuais do protocolo MQTT são a MQTT v3.1.1 e a MQTT v5.0, ambas padronizadas pelo OASIS. Porém, apenas a versão v3.1.1 também é padrão ISO (ISO, 2022) (Organização Internacional para Padronização, do inglês *International Organization for Standardization*).

Cope (2018) ainda descreve que o protocolo MQTT proporciona ao usuário um modelo leve e eficiente de envio de mensagens, sendo desenvolvido para otimizar o consumo de largura de banda da *Internet*. Por esse motivo, o MQTT tem sido amplamente implementado em aplicações IoT.

Segundo Cope (2018) e OASIS (2014), um servidor de mensagens MQTT funciona como um filtro de mensagens baseado em tópicos, que consiste em apenas uma *string* codificada em UTF-8 que serve para endereçar mensagens para os clientes conectados. Portanto, quando um cliente envia mensagens para o servidor de mensagens, ele também indica para qual tópico essas mensagens são direcionadas. Os clientes, com interesse em consumir mensagens, se subscrevem nos tópicos que desejarem. Portanto, o servidor de mensagens MQTT recebe mensagens dos clientes e as separa por seu respectivo tópico e, por fim, as distribui aos clientes que fizeram subscrição em cada tópico.

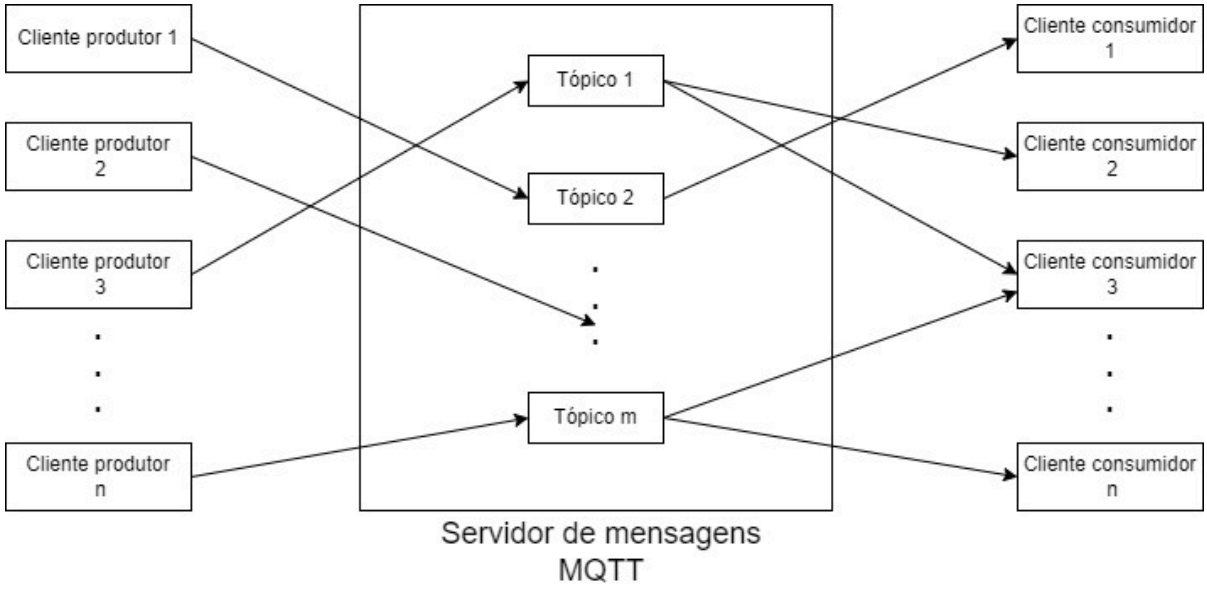
Como exemplo, pode-se citar um sistema IoT em uma casa inteligente, onde um computador monitor se subscreve em um servidor de mensagens MQTT para receber dados dessa casa. Sendo assim, o computador poderia se subscrever em um tópico “/cozinha” para receber mensagens vindas da cozinha, “/lavanderia” para mensagens vindas da lavanderia, “/sala” para mensagens vindas da sala de estar e assim por diante. Neste cenário, os dispositivos IoT dentro da casa enviam mensagens com dados para o servidor de mensagens MQTT. Ou seja, os dispositivos na cozinha poderiam enviar mensagens direcionadas para o tópico “/cozinha”, os dispositivos na lavanderia poderiam enviar mensagens direcionadas para o tópico “/lavanderia” e assim por diante.

É possível criar tópicos e subtópicos no servidor, que podem filtrar melhor a entrega de mensagens para os clientes. Assim, os clientes consumidores podem escolher melhor qual grupo de mensagens tem interesse dentro de um mesmo tópico. Por exemplo, ainda considerando um sistema IoT em uma casa inteligente, um cliente poderia se subscrever nos subtópicos: “/cozinha/temperatura” para receber informações sobre a temperatura da cozinha; “/cozinha/umidade” para receber mensagens sobre a umidade da cozinha; “/cozinha/lampadas” para saber se as lâmpadas foram acesas ou não; Ou ainda, o cliente poderia se subscrever em “/cozinha/#” para receber todas as mensagens enviadas ao tópico “/cozinha” e seus respectivos subtópicos.

Segundo OASIS (2014), uma das principais características do protocolo MQTT são os seus três níveis de qualidade de serviço (QoS, do inglês *Quality of Service*):

- No máximo uma vez (QoS0): neste nível, pode ocorrer de uma mensagem não ser entregue ao consumidor dependendo da conexão entre o cliente e o servidor de mensagens.
- No mínimo uma vez (QoS1): este nível garante que a mensagem seja entregue pelo menos uma vez, o que significa que uma mesma mensagem pode ser entregue mais de uma vez. Neste caso, o consumidor e o servidor de mensagens devem ser capazes de lidar com as mensagens repetidas;
- Apenas uma vez (QoS2): neste nível, o servidor de mensagens garante que a mensagem seja entregue ao consumidor apenas uma vez, mas requer uma troca de quatro pacotes de mensagens, o que reduz o desempenho do servidor de mensagens MQTT.

Figura 2 – Exemplo do funcionamento de um servidor de mensagens MQTT



Fonte: Autoria própria.

O protocolo funciona como na Figura 2. O produtor envia uma mensagem para o servidor de mensagens direcionada para um determinado tópico. Assim, o servidor redireciona a mensagem para todos os consumidores ativos que fizeram subscrição neste tópico.

O protocolo MQTT-SN foi criado para uma rede de sensores e atuadores com poucos recursos computacionais e sem capacidade de oferecer suporte para o protocolo TCP. Para isso, os tópicos são mais simplificados, onde cada tópico permite utilizar apenas um valor de 2 *bytes*. Quanto à sua estrutura, além do servidor de mensagens MQTT-SN, é necessário que *gateways* sejam implementados na rede, já que eles auxiliam na troca de mensagens sem o uso do protocolo TCP (SOUZA, 2018).

2.4.2.2 AMQP

O protocolo de comunicação AMQP foi desenvolvido em 2003 pela empresa JPMorgan para ser utilizado em mercados financeiros. O protocolo foi desenvolvido para ser confiável e seguro, com alto nível de interoperabilidade e provisionamento (NAIK, 2017). Por esse motivo, o AMQP é muito utilizado em sistemas IoT que exigem modelos de comunicação robustos e de fácil utilização. As duas versões atuais deste protocolo são a 1.0 e a 0.9.1.

O AMQP também utiliza o protocolo TCP na camada de transporte. Além disso, o AMQP funciona como um protocolo bidimensional e dá suporte tanto ao modelo publicação/subscrição quanto ao modelo requisição/resposta. Este último modelo é um método síncrono de transmissão, onde o cliente só recebe uma mensagem do servidor se ele enviar uma requisição. Caso mais mensagens estejam disponíveis no servidor no futuro, o cliente só as receberá com novas requisições (KITAMURA, 2021).

RabbitMQ (2011) descreve que em um servidor de mensagens AMQP funcional existe um elemento chamado *exchange*, o qual é utilizado como um tópico em um servidor de mensagens MQTT, porém com mais funcionalidades. Um produtor envia as mensagens para o servidor de mensagens direcionando-as para uma determinada *exchange*. Entretanto, diferentemente dos tópicos, as mensagens não são redirecionadas diretamente para os consumidores, mas para filas contidas no próprio servidor de mensagens AMQP. No processo de distribuição de mensagens dentro do servidor, ainda podem ser usadas chaves de roteamento, que consistem em uma *string* enviada junto com a mensagem com a finalidade de indicar o destino desejado.

De acordo com RabbitMQ (2011), os tipos de *exchanges* podem ser:

- *Default exchange*: neste tipo existe apenas uma *exchange* e todas as filas criadas no servidor de mensagens são associadas a ela. As chaves de roteamento são usadas pela *exchange* para distribuir as mensagens para as filas. Uma mensagem só é enviada para uma determinada fila se a chave de roteamento contiver o nome desta mesma fila. Por exemplo, se existe uma fila com o nome “FilaA”, uma mensagem só será enviada para ela se o valor da sua chave de roteamento for “FilaA”;

- *Direct exchange*: neste tipo, uma mensagem deve ser enviada juntamente com uma chave de roteamento. Um vínculo é criado entre a *exchange* e uma ou mais filas, mas com base em um mapeamento que utiliza as chaves de roteamento para definir os destinos das mensagens. Por exemplo, ao criar a fila “FilaA” e associá-la ao *exchange* “ExA”, também é indicado que apenas as mensagens cujo o valor da chave de roteamento sejam “rota.chaveA” podem ser enviadas para a “FilaA”. Caso contrário, as mensagens não serão enviadas para a fila “FilaA”, mas sim para uma rota padrão;

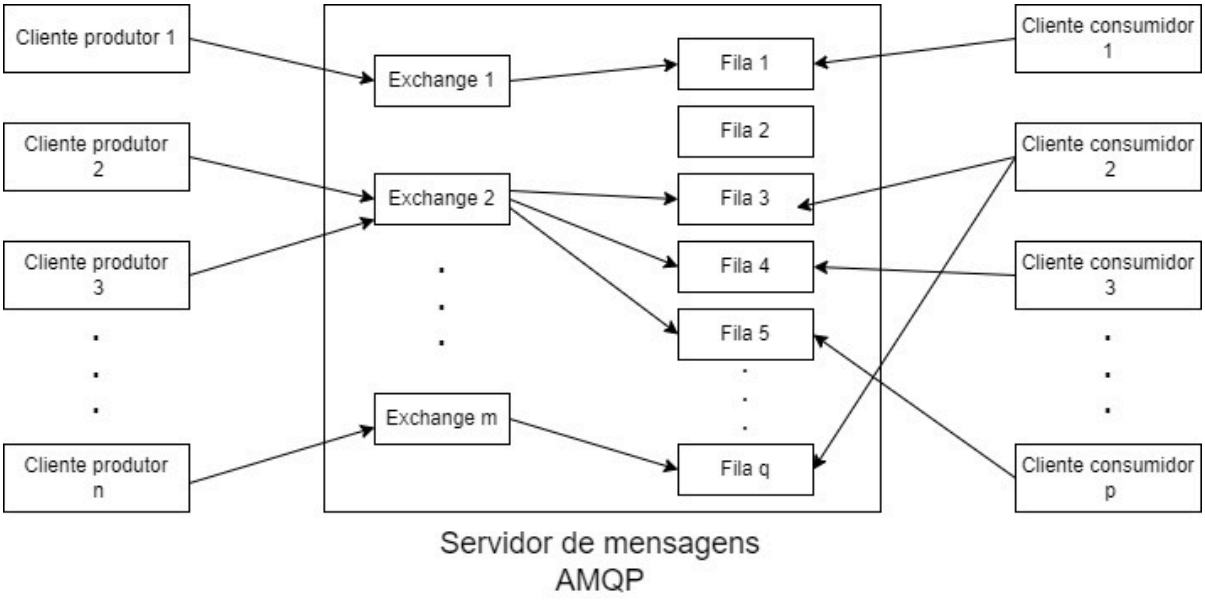
- *Fanout exchange*: neste caso, as mensagens recebidas por uma *exchange* são enviadas para todas as filas que estão associadas a ela, independente dos valores das chaves de roteamento enviadas junto com as mensagens;

- *Topic exchange*: neste tipo, o roteamento de mensagem entre uma *exchange* e as filas é baseado na combinação do valor da chave de roteamento e outras características da mensagem, como, por exemplo, o tipo ou tamanho;
- *Header exchange*: aqui o roteamento de mensagens entre a *exchange* e as filas é feito a partir dos valores presentes nos cabeçalhos das mensagens. Neste caso, o valor presente na chave de roteamento é ignorado.

Apenas a *Direct exchange* é utilizada neste trabalho. Essa escolha foi feita porque este tipo de *exchange* contém mais opções de configuração que a *Default exchange*, mas ainda mantém a característica de que o destino das mensagens em uma *exchange* é definido exclusivamente pelas chaves de roteamento. Este não seria o caso se fossem utilizadas as *Header exchange*, *Topic exchange* e *Fanout exchange*.

A Figura 3 mostra um exemplo de como funciona o protocolo AMQP com a *Direct exchange*. O produtor envia uma mensagem para o servidor de mensagens direcionando-as para uma *Exchange*. Então, a mensagem é encaminhada para uma ou mais filas de acordo com os vínculos estabelecidos no servidor. Os consumidores, por sua vez, consomem as mensagens das filas.

Figura 3 – Exemplo do funcionamento de um servidor de mensagens AMQP



Fonte: Autoria própria.

2.4.2.3 STOMP

STOMP é um protocolo leve de comunicação assíncrona que, como os dois protocolos descritos anteriormente, usa o modelo publicação/subscrição. Nesse protocolo, a comunicação entre os clientes é intermediada por um servidor de mensagens (MIMOUNI; BOUHDADI, 2015).

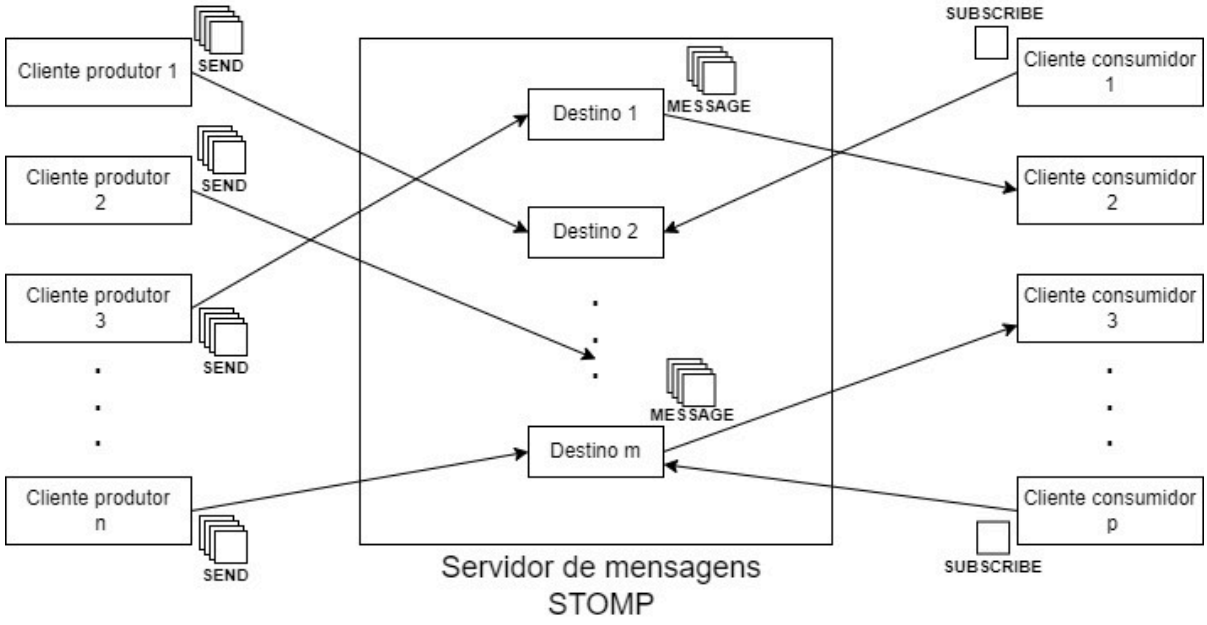
Sua versão atual é a 1.2, a qual pode ser utilizada em vários servidores de mensagens do mercado.

Segundo STOMP (2013), o protocolo STOMP foi criado para ter facilidade de uso e compreensão, com a intenção de melhorar a conexão de servidores de mensagens feita a partir de linguagens de *scripting*, como o Ruby e o Python. Mesmo com o seu desenvolvimento ao longo dos anos, o STOMP continua sendo fiel aos seus princípios iniciais de simplicidade e interoperabilidade. Isto resultou em um protocolo com um menor conjunto de operações disponíveis para o envio de mensagens.

Como descreve STOMP (2013), o protocolo STOMP é baseado no uso de quadros, criados em um formato parecido com o HTTP (Protocolo de Transferência de Hipertexto, do inglês *Hypertext Transfer Protocol*). Cada quadro contém um cabeçalho, um conjunto de valores opcionais e o restante é o conteúdo da mensagem. Portanto, cada mensagem é dividida em um conjunto de quadros antes de ser enviada do produtor para o servidor de mensagens e, depois, todos os quadros são enviados sequencialmente para os consumidores.

O destino da mensagem no servidor de mensagens é apresentado em uma *string* que é enviada junto com a mensagem. Entretanto, o STOMP não define a semântica dessa *string* diretamente, o que deve ser feito pelo próprio servidor de mensagens que implementa este protocolo (STOMP, 2013). Isto permite que a dinâmica do repasse de mensagens seja feita de forma exclusiva, o que pode melhor atender a determinadas necessidades dos usuários. Entretanto, essa liberdade de semântica também reduz a interoperabilidade entre diferentes tipos de servidores de mensagens.

Figura 4 – Exemplo do funcionamento de um servidor de mensagens STOMP



Fonte: Autoria própria.

A Figura 4 mostra um exemplo de como funciona o protocolo STOMP. Um cliente produtor publica mensagens no servidor de mensagens por meio de *frames* do tipo SEND, indicando

um destino. Um cliente consumidor realiza uma subscrição ao enviar um *frame* do tipo SUBSCRIBE, indicando o destino do servidor de mensagens ao qual quer se subscrever. Assim, os clientes que realizaram subscrições recebem as mensagens do servidor por meio de *frames* do tipo MESSAGE.

2.5 Trabalhos relacionados

Nesta seção são apresentados alguns trabalhos disponíveis na literatura cujos objetivos e/ou objetos de estudos são semelhantes aos do presente trabalho.

Luzuriaga *et al.* (2015) fez uma comparação entre o desempenho dos protocolos AMQP e MQTT em relação à perda de mensagens, latência, perturbações na transmissão e valores de limite de saturação nas mensagens (limite de carga de trabalho). As trocas de mensagens foram feitas em redes instáveis com computação móvel. A partir dos resultados encontrados, as conclusões foram que os critérios mais importantes na escolha entre os protocolos devem estar relacionados à segurança e eficiência de energia. Neste caso, o protocolo AMQP seria mais robusto quanto ao critério de segurança e deveria ser implementado em infraestruturas de conglomerados de mensagens que sejam confiáveis e escaláveis. Já o protocolo MQTT seria mais eficiente quanto ao uso de energia, podendo ser implementado como um protocolo de suporte para comunicação com os nós de borda.

Naik (2017) fez uma análise do funcionamento dos protocolos MQTT, COaP, AMQP e HTTP, com a proposta de entender quais deles seriam melhor empregados em algumas métricas de desempenho relacionadas a sistemas IoT. As métricas analisadas foram o Tamanho de mensagens vs Sobrecarga de mensagem, Consumo de energia vs. Requisito de recursos, Largura de banda vs latência, Confiabilidade/QoS vs. Interoperabilidade, Segurança vs. Provisionamento e Uso de M2M (Máquina-para-Máquina, do inglês *Machine-to-Machine*) / IoT vs. Padronização (aferição de qualidade do protocolo feita por organizações de padronização). Vale ressaltar que este estudo é uma análise relativa dos protocolos a partir de outros trabalhos já publicados. Também é importante pontuar que esta análise não considera situações onde a comunicação se dá em condições dinâmicas de redes e com retransmissão de pacotes. Os resultados das análises foram: em relação ao Tamanho de mensagem vs Sobrecarga de mensagem, Consumo de energia vs. Requisito de recursos e Largura de banda vs latência, o CoAP obteve os melhores resultados, seguido pelo MQTT, AMQP e HTTP; em relação à Confiabilidade/QoS vs. Interoperabilidade, o MQTT obteve os melhores resultados para a Confiabilidade/QoS, seguido pelo AMQP, CoAP e HTTP. Porém, para a métrica de Interoperabilidade a ordem é exatamente oposta, com os melhores resultados sendo do HTTP, seguido por CoAP, AMQP e MQTT; em relação à Segurança vs. Provisionamento, o protocolo AMQP obteve os melhores resultados, seguido pelo HTTP, CoAP e MQTT; Por fim, quanto à M2M/IoT vs. Padronização, para M2M/IoT o MQTT obteve os melhores resultados, seguido pelo AMQP, CoAP e HTTP. Já para a Padronização a ordem de melhores resultados é o oposto de M2M/IoT.

Johnsen (2018) fez uma análise sobre o uso do modelo publicação/subscrição para dados IoT de vida curta. O intuito era descobrir os protocolos mais adequados para este tipo de situação, considerando que os dados devem ser entregues o mais rápido possível. Os protocolos testados foram: AMQP, MQTT, MQTT-SN, STOMP, WSN e XMPP. Considerando o tempo de entrega das mensagens, os autores identificaram quais protocolos são mais recomendados para os seguintes requisitos: caso seja exigido o menor tempo de entrega possível, o protocolo mais recomendado é o STOMP, tanto para o produtor quanto para o consumidor; caso o uso do protocolo UDP seja exigido, o protocolo MQTT-SN é mais recomendado tanto para cliente produtor quanto para o consumidor; Caso seja exigido interoperabilidade com sistemas NATO (Organização do Tratado do Atlântico Norte, do inglês *North Atlantic Treaty Organization*), é recomendado que o cliente consumidor utilize o protocolo WSN, enquanto o cliente produtor pode utilizar tanto o MQTT quanto o MQTT-SN; caso seja necessária a criação de uma malha com diferentes tipos de servidores de mensagens, então é recomendado o protocolo AMQP ou o MQTT, tanto para clientes produtores quanto para consumidores; o protocolo XMPP, por sua vez, foi o protocolo que obteve os piores resultados em todos os requisitos avaliados.

Uy e Nam (2019) avaliaram o desempenho dos protocolos AMQP e MQTT por meio de experimentos, simulando latências variadas e taxas de perda de pacotes. Os experimentos foram feitos em uma rede composta de um computador que atuou como cliente consumidor e produtor, outro computador que atuou como servidor de mensagens e um roteador que serviu para estabelecer uma conexão entre os dois computadores. Este estudo foi realizado pelo interesse dos autores na revolução da indústria 4.0 associada a sistemas IoT, que geralmente utilizam os dois protocolos avaliados. Os resultados indicaram que o protocolo MQTT é mais recomendado para dispositivos que tenham restrições de energia, ou para sistemas onde os pacotes não são enviados continuamente e haja baixa taxa de perda de pacotes, até 10%. O protocolo AMQP, por sua vez, tem seu uso recomendado se as mensagens são enviadas continuamente, desde que não haja perda de pacotes ou se a taxa de perda de pacotes for alta; Caso os pacotes não sejam enviados continuamente, seu uso é recomendado se a taxa de perda de pacotes é muito baixa (pela baixa latência que o AMQP apresenta para a transmissão das mensagens) ou então muito alta (já que o AMQP garante que 100% das mensagens são transmitidas, pelo uso de filas).

Ross (2019) analisou o desempenho dos protocolos MQTT e CoAP em dois cenários: monitoramento de sistemas de reservatórios de água; monitoramento da qualidade do ar em cidades. Estes dois cenários foram escolhidos por se adequarem ao funcionamento de sistemas IoT. Os protocolos foram avaliados em relação às seguintes métricas de desempenho: latência, largura de banda, eficiência em condições de perda de dados e consumo de energia. A conclusão da análise foi que o protocolo CoAP obteve os melhores resultados em consideração ao MQTT para os dois cenários propostos.

Bezerra e Westphall (2020) analisou o desempenho dos protocolos AMQP, MQTT e STOMP no ambiente de computação em névoa voltado ao agronegócio. O objetivo foi avaliar

os três protocolos comparando o tempo necessário para realizar a conexão com autenticação, além de avaliar o tempo para completar as trocas de mensagens. Nas comparações foram utilizados diferentes tipos de configurações de segurança. Neste estudo ocorreram diferenças notáveis entre os protocolos, sendo que os resultados foram: considerando o tempo necessário para realizar a conexão com autenticação, o MQTT obteve os melhores resultados, seguido pelo STOMP e AMQP; quanto ao tempo para completar as trocas de mensagens, o protocolo que obteve os melhores resultados foi STOMP, seguido pelo AMQP e MQTT. Os autores concluíram que o protocolo STOMP seria a melhor opção para o cenário proposto, devido às suas opções de segurança e seu desempenho geral quanto às trocas de mensagens.

Wytrebowicz, Cabaj e Krawiec (2021) fizeram uma análise qualitativa e pragmática de alguns protocolos de comunicação, com o foco em seu uso em sistemas IoT. Os protocolos comparados foram MQTT, MQTT-SN, CoAP, STOMP, XMPP, WAMP, AMQP, DDS, OPC UA, LwM2M. As conclusões dos autores foram: para dispositivos IoT com recursos limitados, os protocolos MQTT e CoAP são mais interessantes; para dispositivos que não permitem o uso do protocolo TCP, os únicos disponíveis são o CoAP, MQTT-SN, DDS, e OPC UA; para sistemas IoT simples, sem limitações de recursos, os protocolos STOMP e WebSocket podem ser uma escolha mais otimizada; para uma grande quantidade de recursos de gerenciamento de mensagens, os mais recomendados são LwM2M, AMQP, DDS, e OPC UA; por fim, os protocolos DDS e OPC UA, sendo os mais complexos entre os analisados, são os mais recomendados para grandes sistemas IoT.

Existe um crescente uso de dispositivos móveis, os quais dispõem de recursos limitados em sistemas IoT e, por essa razão, podem exigir o uso de computação em nuvem e computação em névoa. Diante desse cenário, o presente trabalho propõe a realização de uma análise do desempenho de protocolos de comunicação AMQP, MQTT e STOMP, que são muito utilizados em sistemas IoT, em um dispositivo móvel. O próximo capítulo descreve o projeto de software criado neste trabalho.

3 PROJETO DE SOFTWARE

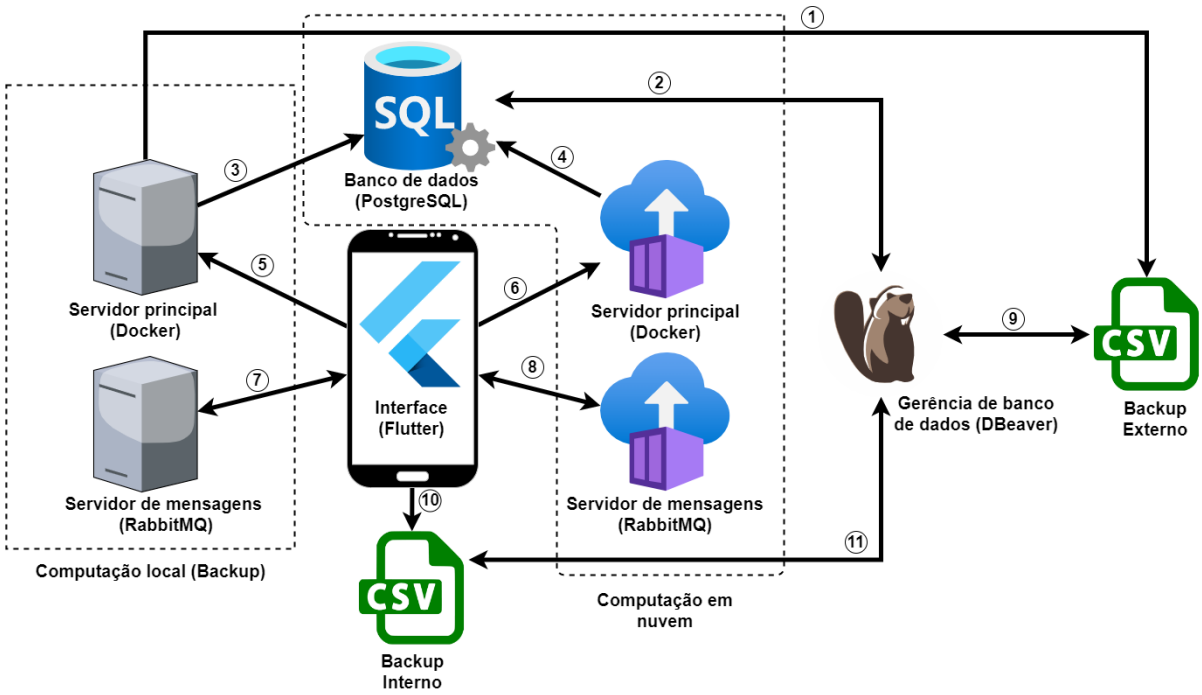
Este capítulo inicia com a apresentação da arquitetura de software proposta (Seção 3.1), com descrições das tecnologias (Seção 3.2) e linguagens de programação utilizadas (Seção 3.3). Em seguida, na Seção 3.4, são apresentadas as características de funcionamento da aplicação criada para realizar a análise do dispositivo móvel, ao utilizar os protocolos AMQP, MQTT e STOMP. Portanto, é detalhado o ciclo de vida de um sistema operacional Android (Seção 3.4.1), são ilustrados o diagrama de classes (Seção 3.4.2) e as telas da aplicação criada para o dispositivo móvel (Seção 3.4.3).

3.1 Arquitetura de Software

A Figura 5 apresenta a arquitetura do sistema, evidenciando as relações entre as tecnologias utilizadas.

O dispositivo móvel é o ponto central da aplicação, não só porque ele faz parte da comunicação que será analisada, mas também porque a coordena. A troca de mensagens ocorre entre o dispositivo móvel e um servidor de mensagens, que funciona como um nó de névoa. Essa comunicação é representada pela seta de numeração 8.

Figura 5 – Arquitetura de Software



Fonte: Autoria própria.

Os dados de comunicação ficam temporariamente armazenados na memória interna do dispositivo móvel, em um arquivo no formato CSV (indicado pela seta de número 10). Estes dados armazenados podem ser enviados para um banco de dados externo, que é executado como um microsserviço Docker (INC, 2022) no servidor de computação em nuvem Heroku

(HEROKU, 2022). Neste caso, os dados do arquivo CSV são transmitidos para o “Servidor Principal” (indicado na seta número 6) e, posteriormente, são salvos em um Banco de dados PostgreSQL (GROUP, 2022), indicado pela seta de número 4.

Caso haja problema em estabelecer a conexão entre o dispositivo móvel e os microsserviços, uma instância local pode ser criada e configurada localmente para permitir que experimentos sejam realizados. Neste caso, as conexões representadas pelos números 7, 5 e 3 substituem as conexões 8, 6 e 4, respectivamente.

Como forma de redundância, um segundo arquivo CSV, gerado a partir do banco de dados, é periodicamente armazenado em outro servidor na nuvem. Esta conexão é representada pelo número 9. Em caso de problemas de conexão com a instância principal do servidor, tratada anteriormente, a conexão 9 é substituída pela conexão de número 1.

Os dados salvos no banco de dados e nos dois arquivos CSV são sincronizados pela ferramenta DBeaver (TEAM, 2021). As conexões do DBeaver são representadas pelos números 11, 9 e 2. O DBeaver também é utilizado para gerenciar os dados dos experimentos para análise.

A seguir são detalhadas as tecnologias utilizadas.

3.2 Tecnologias utilizadas

O RabbitMQ (RABBITMQ, 2022a) foi escolhido como o servidor de mensagens para a implementação deste trabalho pelo fato dele oferecer suporte aos três protocolos analisados (MQTT, STOMP e AMQP) de forma nativa. O RabbitMQ é um servidor de mensagens de código aberto muito utilizado no mundo e que permite o uso de vários protocolos de mensagens. Ele é leve, de fácil uso, pode ser usado em vários sistemas operacionais e tem suporte de bibliotecas para várias linguagens de programação (RABBITMQ, 2022a).

Em um servidor de mensagens RabbitMQ, pode-se utilizar de forma oficial os seguintes protocolos, quando devidamente configurados (RABBITMQ, 2022b):

- AMQP: tanto a versão 0-9-1 como a versão 1.0 podem ser utilizadas;
- STOMP: todas as versões existentes podem ser utilizadas;
- MQTT: o RabbitMQ dá suporte apenas para a versão 3.1.1;
- WebSockets: o servidor de mensagens pode enviar mensagens para navegadores de *Internet* usando WebSockets.

Para o banco de dados escolheu-se o PostgreSQL (GROUP, 2022), por ser uma ferramenta robusta e pela familiaridade que os membros da equipe já possuem com ela. O PostgreSQL é um sistema de banco de dados relacional de código aberto, há mais de 30 anos em

desenvolvimento e com suporte para o padrão SQL. O PostgreSQL é reconhecido por ser robusto, confiável e com um ótimo desempenho. Foi desenvolvido a partir do POSTGRES, criado na Universidade da Califórnia, que foi pioneiro em muitos conceitos (GROUP, 2022).

Para serviços em nuvem foi escolhida a plataforma Heroku (HEROKU, 2022) para o servidor principal, enquanto uma instância local foi escolhida para o servidor de mensagens. Heroku é uma empresa que oferece serviços do tipo PaaS na nuvem, sendo uma das mais antigas nesse tipo de serviço (HEROKU, 2022). Entre alguns serviços importantes que ela disponibiliza está a possibilidade de implementar em seus servidores aplicações completas. O banco de dados utilizado neste projeto foi lançado por meio da Heroku.

Docker é uma plataforma aberta que tem como objetivo auxiliar desenvolvedores na produção, distribuição e execução de aplicações criadas por eles, já que ele permite ao usuário criar imagens executáveis (INC, 2022). Ao usar o Docker, o desenvolvedor cria um ambiente chamado de contêiner que, de acordo com Inc (2022), é usado pelo usuário para empacotar sua aplicação para distribuição futura, sendo que contêineres possuem todos os recursos necessários para que a aplicação seja executada adequadamente em diferentes tipos de ambientes, por não exigir o uso de dependências externas ao contêiner. O Docker é utilizado neste trabalho para criar as imagens executáveis do Servidor Principal e do Servidor de Mensagens, para execução nas plataformas de serviço em nuvem e também nas instâncias locais.

DBeaver é uma ferramenta de administração de banco de dados universal, de uso livre, multiplataforma e de código aberto (TEAM, 2021). Para isso, atualmente, o DBeaver tem suporte para vários tipos de bancos de dados, entre os mais famosos estão: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird e Apache Hive. O DBeaver permite ao usuário editar o conteúdo de bancos de dados, transferir ou copiar os dados para arquivos locais. Esta ferramenta é utilizada para gerenciar os dados gerados a partir dos experimentos para análise do dispositivo móvel, realizados durante a produção deste trabalho (descritos em detalhes no Capítulo 4). O DBeaver é utilizado no gerenciamento de dados armazenados no banco de dados e dos arquivos CSV.

Flutter é uma ferramenta criada para o desenvolvimento de aplicativos voltados para dispositivos móveis, com uma compilação feita de forma nativa para os sistemas operacionais Android, iOS, Windows, Mac, Linux, Fuchsia e Web (GOOGLE, 2022). O Flutter foi criado em 2015 pela Google com um código aberto e é baseado na linguagem de programação Dart (DART, 2022).

3.3 Linguagens de Programação

Nesta seção estão descritas as linguagens de programação utilizadas na implementação deste projeto e algumas bibliotecas importantes para o funcionamento do sistema.

Como descrito em America (2021), Python é uma linguagem muito utilizada desde a produção de pequenas aplicações até projetos de maior porte envolvendo ciência de dados,

que permite a criação de bibliotecas nativas de forma fácil e com uma baixa curva de aprendizado. Estas mesmas razões fizeram com que o Python fosse utilizado neste projeto, mais especificamente na implementação do Servidor Principal.

A seguir estão descritas algumas das bibliotecas para a linguagem Python que são utilizadas no presente trabalho:

- Flask: utilizada para interfacear as requisições REST (*Representational State Transfer*) entre o dispositivo móvel e o servidor principal. Esta biblioteca é um *framework* de Interface de Porta de Entrada do Servidor Web para Python, que serve para interfacear servidores web e aplicações web, ou frameworks (FLASK, 2022);
- Psycopg: permite que um banco de dados PostgreSQL seja utilizado em Python. Por ter sido desenvolvido em C, esta biblioteca é eficiente e segura. Ela suporta que mais de uma *thread* compartilhe uma mesma conexão e que a comunicação seja feita de forma assíncrona (PSYCOPG2, 2022).

Dart é uma linguagem de programação livre e de código aberto criada com suporte da Google (DART, 2022). Ela se distingue por ser otimizada para criação de interfaces de usuário e de fácil uso, além de ter suporte para vários tipos de processadores. A Dart ainda pode ser compilada para executar JavaScript em navegadores web (DART, 2022). Esta linguagem é utilizada neste projeto no desenvolvimento de aplicativos para os dispositivos móveis por meio da plataforma Flutter.

Algumas das bibliotecas Dart utilizadas são:

- dart_ampq é um cliente AMQP (DART_AMQP, 2021). É utilizada para fazer a conexão com o servidor de mensagens, além de enviar e receber as mensagens;
- mqtt_client é um cliente MQTT para Dart que funciona para estabelecer comunicação com os navegadores de *Internet* e com os servidores de mensagens (MQTT_CLIENT, 2022). É utilizada para fazer a troca de mensagens com servidor de mensagens utilizando o protocolo MQTT;
- stomp.dart é um cliente STOMP para Dart que é utilizado para realizar as trocas de mensagens com o servidor de mensagens (STOMP.DART, 2019);
- http.dart é uma biblioteca dart que permite fazer requisições HTTP. É utilizada para enviar os dados de experimentos do dispositivo móvel para o servidor principal, para serem salvos no banco de dados (HTTP.DART, 2021).

SQL é uma linguagem muito utilizada para extrair informações de bancos de dados relacionais por meio de consultas (SILVEIRA, 2019). SQL é uma linguagem declarativa que, segundo America (2021), é capaz de executar tanto ações transacionais quanto analíticas, e

ao ser implementada possibilita a padronização do banco de dados, acesso direto aos dados, além de maior flexibilidade e simplicidade no uso de tecnologias. Esta linguagem é utilizada no banco de dados PostgreSQL do projeto.

O MATLAB é uma plataforma para programação e computação numérica robusta e muito utilizada ao redor do mundo (MATHWORKS, 2022). Pelo fato de permitir vários tipos de cálculos matemáticos complexos e produzir gráficos, foi escolhida para fazer a análise dos dados e gerar os gráficos apresentados no Capítulo 4.

3.4 Dispositivo Móvel

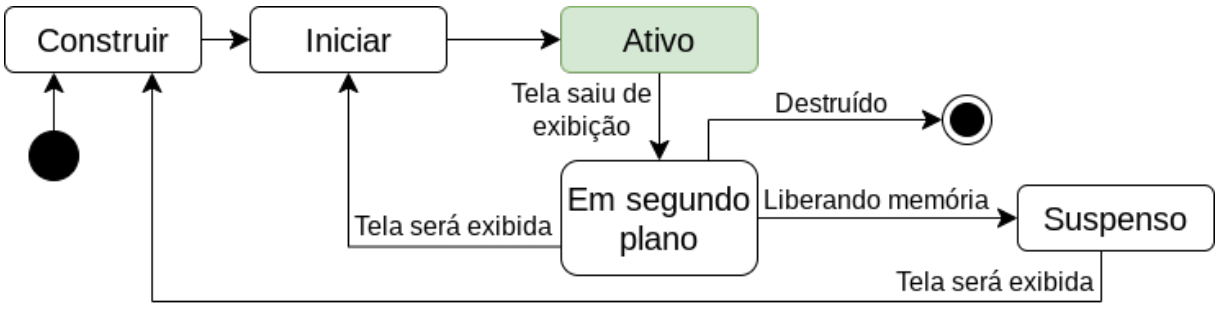
Nesta seção estão descritas algumas características importantes para o funcionamento da aplicação criada para realizar a análise do dispositivo móvel utilizado. Essas características devem ser levadas em consideração para que a aplicação funcione de acordo com o esperado e que os experimentos não sofram com interferências indesejadas.

O aplicativo é produzido para ser executado em um dispositivo móvel com o sistema operacional Android. O entendimento do ciclo de vida de um sistema operacional Android é importante para evitar qualquer tipo de comportamento que prejudique a análise do dispositivo móvel. Este ciclo é detalhado a seguir.

3.4.1 Diagrama de ciclo de vida Android

Esta seção apresenta o diagrama simplificado de ciclo de vida do aplicativo para o sistema operacional Android.

Figura 6 – Diagrama de ciclo de vida de um aplicativo Android.



Fonte: Autoria própria.

A Figura 6 apresenta o ciclo de vida de uma tela de um aplicativo Android. Ao iniciar uma aplicação, existe uma série de métodos que são acionados como preparação inicial, antes da tela do aplicativo ser apresentada para o usuário. Neste momento, o aplicativo está no estado de “Construir”.

No estado “Iniciar”, apesar da tela do aplicativo estar pronta para visualização, ela ainda não está sendo apresentada para o usuário. Esse é um estado de transição que pode acontecer de duas formas: caso a tela tenha tido sua construção finalizada, mas ainda não foi apresentada ao usuário; caso a tela perca seu estado de Ativo e esteja no estado “Em segundo plano” , mas está na iminência de ser exibida novamente.

No estado “Ativo”, a tela da aplicação está sendo exibida ao usuário no visor do dispositivo móvel e é o único momento em que a aplicação tem acesso a todos os recursos do dispositivo. Em qualquer outro estado apresentado, o sistema não garante que os dados produzidos sejam mantidos em memória.

Quando ocorre a transição de “Ativo” para “Em segundo plano” a tela do aplicativo fica inoperante, suas atividades são completamente interrompidas e seus dados ficam salvos em memória. A tela da aplicação permanece neste estado até que ela tenha a oportunidade de retornar às suas atividades. Entretanto, caso uma outra aplicação de maior prioridade precise usar mais memória do dispositivo móvel, que não esteja livre, então a tela da aplicação no estado “Em Segundo Plano” é destruída pelo Sistema Operacional, apagando inclusive seus dados salvos em memória. Quando esta situação ocorre, a tela destruída passa do estado “Em Segundo Plano” para o estado de “Suspenso”.

Caso uma tela no estado de “Suspenso” precise ser utilizada novamente no dispositivo, ela passa para o estado de “Construir”, onde todos os recursos necessários são alocados novamente, porém nenhum dado produzido anteriormente será recuperado pela aplicação.

Existe uma alternativa para manter os dados das telas da aplicação salvos em memória mesmo se ela estiver no estado de “Suspenso”, que consiste no uso do componente *ViewModel*, do sistema Android, apresentado na Seção 3.4.2. O *ViewModel* é um componente capaz de persistir os dados que foram salvos nele até que o aplicativo seja finalizado.

Existe um problema que pode ser gerado quando a aplicação não está mais no estado “Ativo”, pois o sistema operacional pode negar à aplicação o acesso ao adaptador de rede. Neste caso, o efeito para o aplicativo é o mesmo de perder conexão com a *Internet*.

3.4.2 Diagrama de classes

Nesta seção, é apresentado um diagrama de classes da aplicação criada para o dispositivo móvel, o qual segue uma adaptação da arquitetura MVVM (*Model-View-ViewModel*), com a adição de uma camada adicional de repositório que serve para desacoplar o armazenamento de dados da lógica de negócio da aplicação.

As camadas do modelo adotado são:

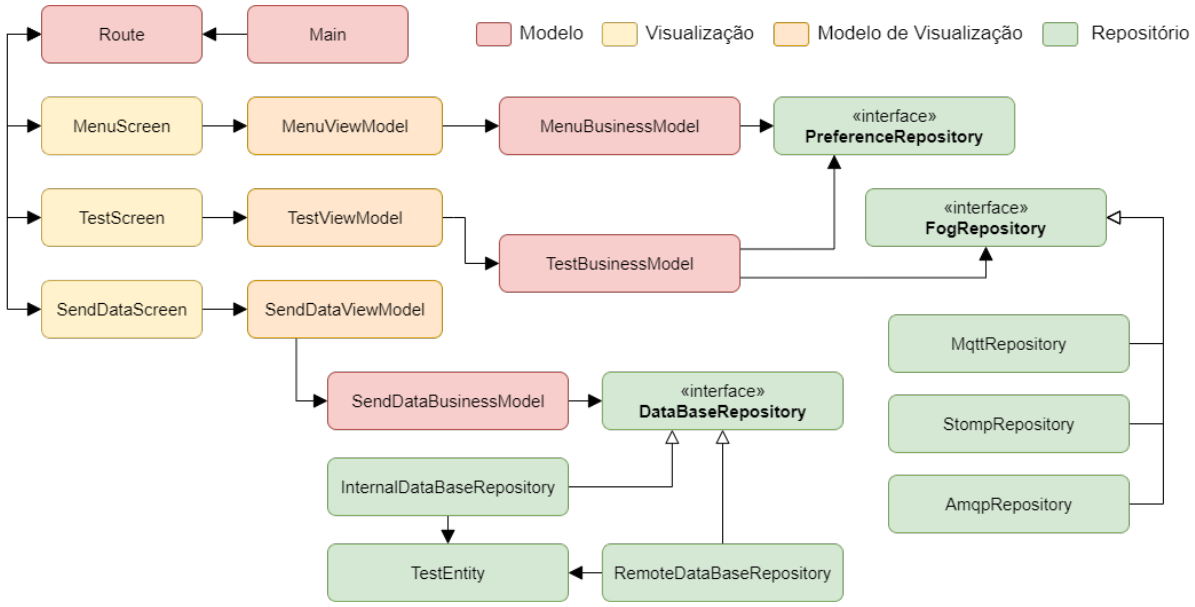
- Visualização: é responsável por exibir informações para o usuário e capturar as ações do mesmo;

- Modelo de Visualização: é responsável por fazer o mapeamento entre o formato de informação exibida para o usuário e o formato que o modelo consegue tratar. Também faz o mapeamento das ações do usuário para casos de uso do modelo;
- Modelo: é responsável pelo processamento de dados e pela lógica de negócio;
- Repositório: é responsável por persistir dados e pela comunicação com serviços externos ou internos.

A Figura 7 apresenta o diagrama de classes do projeto onde as classes de Modelo estão em Rosa, a Visualização em Amarelo, o Modelo de Visualização em laranja e o Repositório em verde.

Como a aplicação tem três telas, existem classes associadas diretamente a cada uma delas. Cada uma das telas está apresentada na Seção 3.4.3.

Figura 7 – Diagrama de classes da aplicação do dispositivo móvel



Fonte: Autoria própria.

A tela de menu é representada pela classe *MenuScreen* que é responsável por apresentar a interface do usuário no visor do dispositivo móvel, sem fazer nenhum tipo de armazenamento de dados. Ela também é usada para receber os valores configurados pelo usuário para a realização dos experimentos que são apresentados em detalhes no Capítulo 4.

A *MenuViewModel* serve para preparar os dados salvos no dispositivo para a *MenuScreen* apresentar ao usuário e para disponibilizar as opções de experimento configuradas pelo usuário para a classe *MenuBusinessModel*.

Já a *MenuBusinessModel* é a classe que faz todo gerenciamento dos dados das classes da tela de Menu. No modelo MVVM, apenas as classes do tipo Modelo têm acesso aos recursos do dispositivo.

A tela de experimento é representada pela classe *TestScreen* que indica ao usuário que o experimento está sendo realizado e permite interrompê-lo caso seja necessário.

A classe *TestViewModel* envia o comando para que a classe *TestBusinessModel* inicie os experimentos, mas também indica para a *TestBusinessModel* se o usuário decidir interromper o processo. Para a realização dos experimentos, os dados utilizados na classe *TestBusinessModel* são obtidos da *MenuBusinessModel* e não pelas outras classes associadas ao experimento.

A *FogRepository* é uma classe abstrata que está associada à *TestBusinessModel*. Ela contém os métodos para estabelecer conexão com o servidor de mensagens e, também, serve como um contrato para que a implementação efetiva do protocolo de comunicação possa gerenciar toda a conexão com o servidor de mensagens, sem que a *TestViewModel* precise conhecer essa implementação. A classe *FogRepository* representa três classes ligadas ao envio de mensagens para o servidor de mensagens, cada uma delas utiliza um dos protocolos implementados neste projeto. Portanto, *MqttRepository* transmite mensagens através do protocolo MQTT, *StompRepository* transmite mensagens através do protocolo Stomp e *AmqpRepository* transmite mensagens através do protocolo AMQP. Essa abordagem permite que novas implementações de protocolos de comunicação sejam testadas em projetos futuros.

A última tela apresentada ao usuário é a de envio de dados que é representada pela classe *SendDataScreen*. Essa tela apresenta ao usuário a possibilidade de enviar ao servidor de mensagens os dados dos experimentos. Então, a *SendDataViewModel* envia para a *SendDataBusinessModel* as opções configuradas pelo usuário. Já a *SendDataBusinessModel* faz o processamento necessário da informação, recuperando os dados dos experimentos e, caso seja necessário, ela aciona a classe *DataBaseRepository*, que contém os métodos responsáveis por enviar os dados dos experimentos do dispositivo móvel para o servidor principal e salvar localmente no arquivo CSV.

A *DataBaseRepository* é uma classe abstrata que representa as classes ligadas ao envio de mensagens ao servidor principal. *RemoteDataBaseRepository* tem a função de transmitir os dados dos experimentos para o servidor principal.

A *InternalDataRepository* é uma classe que é utilizada para armazenar os dados no dispositivo móvel, no arquivo CSV local.

3.4.3 Interface do usuário

Esta seção apresenta as telas do aplicativo executado no dispositivo móvel, conforme pode ser visto na Figura 8. O design utilizado é o mais simples possível, sem qualquer tipo de animação que consuma energia e processamento do dispositivo, com o máximo de cores em tons de cinza para poupar energia e com o mínimo possível de opção para o usuário.

A Figura 8a apresenta a tela inicial do aplicativo que contém as opções de envio de mensagens para os experimentos. As opções disponíveis são:

- Servidor principal: neste campo deve ser colocado o endereço de destino do servidor principal;
- Servidor de mensagens: este campo deve conter o endereço de destino do servidor de mensagens;
- Usuário: neste campo deve ser inserido o usuário que será utilizado no servidor de mensagens, que consiste em uma *string* simples;
- Senha: nesta opção o usuário deve inserir a senha de usuário do servidor de mensagens;
- Opção de tipo de protocolo: existem três opções que podem ser selecionadas e que definem qual protocolo será testado. Apenas uma opção pode ser selecionada por experimento;
- Tamanho da mensagem: nesta opção deve ser inserido um número que indicará o tamanho (em caracteres) do conteúdo que será enviado em cada mensagem para o servidor;
- Quantidade de mensagens: esta opção indica o número de mensagens que devem ser enviadas sequencialmente;
- Intervalo entre mensagens (ms): neste item deve ser indicado o tempo de intervalo entre o envio de mensagens. O valor inserido é considerado em milissegundos;
- Desligar automação: essa caixa de seleção, quando selecionada, faz com que a aplicação realize os experimentos sem a coleta e envio de dados para o servidor principal. Isto é feito para evitar interferências nos dados por parte da aplicação em alguns dos experimentos. Este comportamento será melhor explicado no capítulo 4.

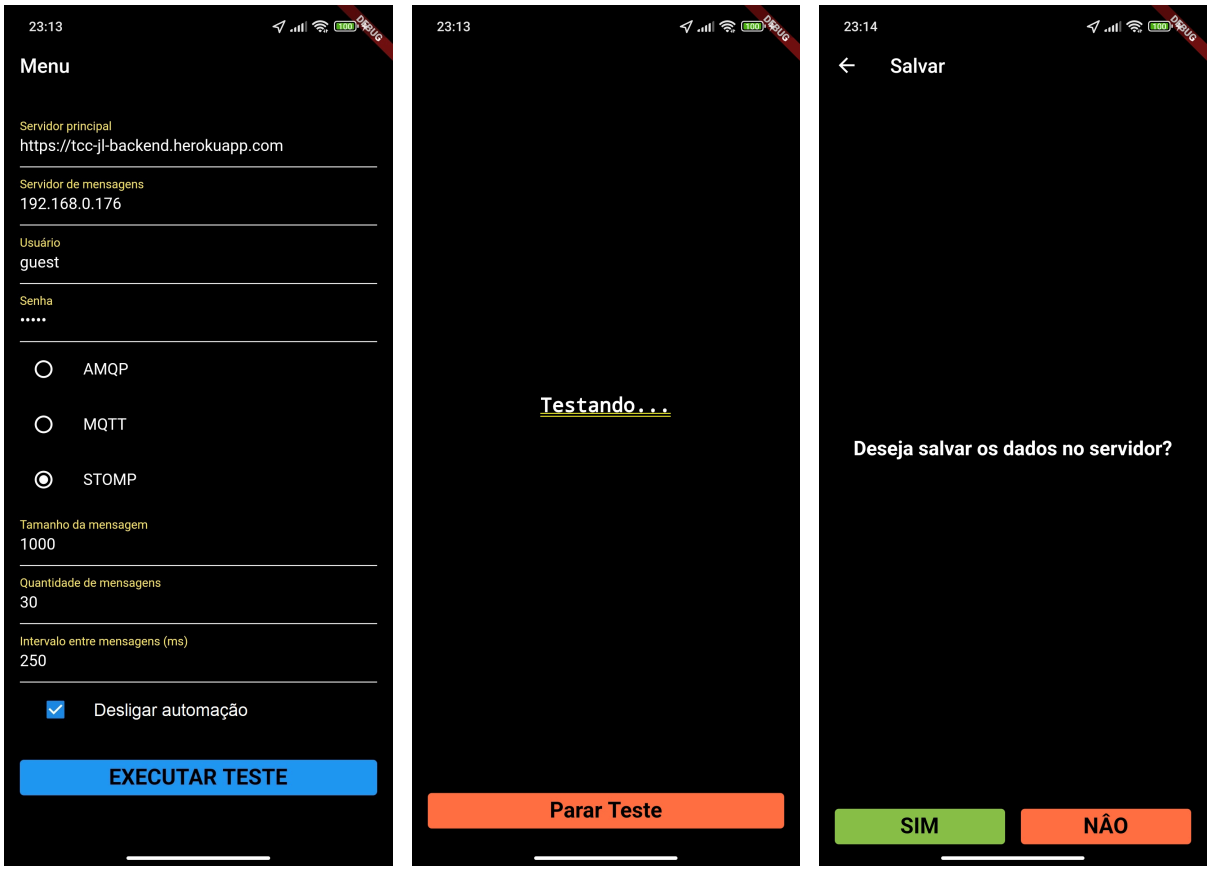
No final da tela existe um botão que inicia o experimento.

A Figura 8b indica ao usuário apenas que o experimento está sendo executado. Ao final da tela existe um botão que permite ao usuário interromper o experimento, neste caso os dados não são salvos.

A Figura 8c apresenta a última tela da aplicação que surge ao final do experimento. Nesta tela existe a opção de enviar ou não as mensagens trocadas para o servidor principal, para serem salvas no banco de dados. As mensagens trocadas que serão enviadas não são apresentadas na tela, já que elas podem ser facilmente extraídas do banco de dados com o uso do DBeaver. Os dados contidos nas mensagens são apresentados no Capítulo 4.

O próximo capítulo apresenta os experimentos utilizados para a análise de desempenho do dispostivo móvel, uma descrição da metodologia empregada, os resultados obtidos com os experimentos e, por fim, uma análise destes resultados.

Figura 8 – Telas do aplicativo



(a) Menu de configuração **(b) Realização do experimento** **(c) Conclusão e confirmação**

Fonte: Autoria própria.

4 ANÁLISE DE DESEMPENHO DOS PROTOCOLOS AMQP, STOMP E MQTT

Este capítulo descreve a metodologia empregada para a execução dos experimentos de análise de desempenho do dispositivo móvel (Seção 4.1) e apresenta os resultados obtidos (Seção 4.3).

4.1 Metodologia

Nesta seção está descrita a metodologia empregada para a execução dos experimentos de análise de desempenho do dispositivo móvel.

O objetivo dos experimentos é analisar o desempenho do dispositivo móvel durante a troca de mensagens com o servidor de mensagens. Mais especificamente, a análise dos experimentos tem como intuito de entender o comportamento do dispositivo móvel quanto às suas limitações que foram apresentadas na Seção 2.1. Para alcançar tal objetivo, as seguintes métricas de desempenho foram estabelecidas:

- Consumo de energia: uma vez que os dispositivos móveis utilizam baterias como fonte de energia, as quais tem limitações, o bom gerenciamento de seu consumo é essencial para o funcionamento. Por esta razão, é muito importante entender como cada protocolo influencia o consumo de energia em um dispositivo móvel;
- Consumo de memória: todo computador tem uma quantidade limitada de memória a sua disposição. Entretanto, dispositivos móveis tendem a ter menos memória disponível para uso. Por esta razão, é interessante entender como cada protocolo influencia o consumo memória em um dispositivo móvel;
- Consumo de processamento: este item tem a mesma implicação do item anterior, porém voltadas ao uso de processador;
- Tempo de ida e volta de mensagem: como os protocolos analisados permitem que as mensagens enviadas possam variar de tamanho, é interessante entender como essa variação pode afetar o tempo de ida e volta de mensagens de cada protocolo no dispositivo móvel;
- Perda de mensagem: os protocolos analisados têm mecanismos para garantir a entrega de mensagens. Por isso, é importante analisar se ocorreram perdas de mensagens durante as trocas realizadas entre o dispositivo móvel e o servidor de mensagens.

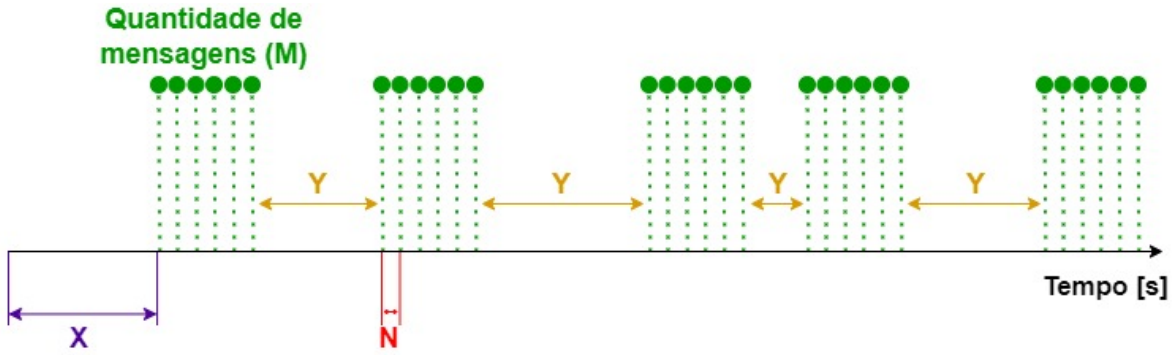
Durante os experimentos, o dispositivo móvel atuou tanto como cliente publicador quanto como cliente consumidor.

A maneira como as mensagens foram trocadas entre o dispositivo móvel e o servidor de mensagens é de extrema importância para a obtenção de resultados significativos. A Figura 9

ilustra um exemplo de envio de mensagens ao longo do tempo durante a execução de um experimento. Os seguintes valores foram configurados nos experimentos:

- O valor X é o tempo utilizado para a estabilização do dispositivo móvel antes do envio das mensagens (para evitar interferências advindas do dispositivo móvel nos dados dos experimentos);
- O valor M é a quantidade de mensagens que são enviadas sequencialmente em um bloco. M é constante durante todo o experimento;
- O valor Y é o tempo de intervalo entre o envio de cada bloco de mensagens. Esse valor é aleatório, variando entre 1 milissegundo e 1 segundo. Essa escolha foi feita para tornar o recebimento de mensagens menos preditivo, o que ajuda a evitar qualquer tipo de otimização na transmissão de dados feita pelo dispositivo móvel. No total são enviados dez blocos de mensagens por experimento;
- O valor N é o tempo de intervalo entre os envios das mensagens em cada bloco enviado.

Figura 9 – Exemplo de envio de mensagens durante os experimentos



Fonte: Autoria própria.

O comportamento do dispositivo móvel durante os experimentos de consumo de energia, uso de memória e uso de processamento é um pouco diferente. Nestes experimentos não são realizados a coleta e armazenamento de dados no próprio dispositivo móvel, já que são utilizadas ferramentas externas para estes fins. Com isso, evitam-se interferências nos dados, que poderiam ocorrer pelo processamento em paralelo dos dados e armazenamento deles no aparelho. Para que os experimentos sejam realizados neste formato, a caixa “Desligar automação” deve estar selecionada na tela do menu de configuração, apresentada na Figura 8a da Seção 3.4.3. Sem a caixa “Desligar automação” selecionada, ao receber uma mensagem do servidor de mensagens, o dispositivo móvel cria uma *thread* para fazer o tratamento adequado dessa mensagem para armazenamento e futuro envio para o servidor principal. Já com a caixa “Desligar automação” selecionada, ao receber uma mensagem do servidor de mensagens, a *thread* citada não é executada, o que evita interferências nos dados referentes aos experimentos de consumo de energia, uso de processamento e consumo de memória.

Os protocolos AMQP e STOMP foram tratados de forma similar no servidor de mensagens: para cada protocolo existiu uma fila exclusiva, onde o dispositivo móvel publicava e de onde ele lia as mensagens. A subscrição foi feita antes do início do envio das mensagens, o que significa que no momento em que a mensagem estava disponível para consumo na fila, o dispositivo móvel já a recebia.

No caso do protocolo MQTT, por conta de seu funcionamento, o servidor de mensagens tinha um Tópico para envio de mensagens. O dispositivo se subscrevia nesse tópico e enviava mensagens para ele também. As mensagens neste protocolo foram enviadas utilizando o QoS1, pois é o máximo que o servidor RabbitMQ permite. Para que o dispositivo móvel recebesse as mensagens dos experimentos corretamente, a subscrição foi feita antes do início do envio das mensagens. Então, assim que a mensagem chegava no servidor ela era enviada para o cliente assinante.

Cada mensagem enviada do dispositivo móvel para o servidor de mensagens continha os dados no formato JSON, com os seguintes campos:

- accessKey: chave de acesso para o servidor principal;
- platform: sistema operacional utilizado para testar os protocolos;
- protocol: protocolo utilizado durante o experimento;
- messageSize: tamanho da mensagem que estava sendo enviada;
- sendTime: hora de envio da mensagem no formato *Timestamp*. Este campo era preenchido no momento em que a mensagem era enviada;
- returnTime: hora de recebimento da mensagem no formato *Timestamp*. Este campo era preenchido assim que a mensagem era recebida pelo dispositivo móvel;
- intervalTime: intervalo de tempo entre o envio do bloco de mensagens anterior e o atual (Valor Y na Figura 9);
- messageDelta: intervalo de tempo entre o envio de cada mensagem dentro de um bloco de mensagens (Valor N na Figura 9).
- messagesSent: número de mensagens enviadas;
- messagesReceived: número de mensagens recebidas;
- x: *string* preenchida com “0” até que a mensagem atinja o número de caracteres desejado.

Para cada métrica analisada, um conjunto de experimentos foi realizado, mas sempre com o mesmo número de amostras entre eles, de acordo com a Figura 9. Em cada série, para cada protocolo, foram realizados 30 experimentos, com 10 mensagens por bloco de mensagens (valor M). Como cada bloco era enviado 10 vezes por experimento, portanto para cada série foram trocadas 3000 mensagens no total.

Cada conjunto de experimentos realizado exigiu o uso de ferramentas diferentes para a coleta de dados.

Para a realização dos experimentos de energia, o dispositivo foi carregado até a totalidade usando como referência o medidor de consumo da marca UNI-T, modelo UT658DUAL. Este medidor permitiu identificar a queda na corrente do carregador durante o processo de carregamento da bateria. A corrente observada durante o processo de carregamento foi de 0,45 A, enquanto a corrente com a bateria plenamente carregada foi de 0,2 A. Ambos os valores de corrente são referentes a uma carga lenta da bateria.

Foi realizada uma tentativa de experimento com carregamento rápido, mas o consumo de energia praticamente dobrou, de modo que neste trabalho foram analisados apenas os dados obtidos com consumo de energia a partir do carregamento lento. Como existe uma diferença entre os tipos de carga, antes dos experimentos o dispositivo foi totalmente descarregado e posteriormente carregado de forma lenta.

Para o experimento de memória foi empregado o analisador de memória do Flutter, o qual faz a comunicação por meio de *websocket* com um navegador (MOZILLA, 2022). Esse experimento causou uma pequena interferência na leitura do consumo de memória, mas foi a única forma de obter essas informações. Esta limitação se deve a restrições que a Google vem impondo quanto ao nível de acesso a informações do sistema apenas para aplicativos autorizados (ANDROID, 2019), sendo a comunicação por *websocket* a ferramenta oficial para esse tipo de análise em tempo de desenvolvimento.

O experimento de processamento foi feito empregando a ferramenta de *debug* de processos do Flutter, que mostra o tempo em que cada processo foi executado. Foi considerado o tempo de execução do processo “handleMessages” do Dart, que é responsável por tratar as mensagens que foram recebidas pelo *socket* do sistema. Neste caso, foram desconsiderados os valores de envios de mensagens para evitar a interferência da ferramenta durante a execução do experimento, já que os dados dessa ferramenta de *debug* são transmitidos por meio de *websockets*.

Já na análise do tempo médio para concluir a transmissão de cada mensagem e da taxa de perda de mensagens, foram utilizados os próprios dados presentes nas mensagens enviadas ao servidor principal. Os dados utilizados foram o horário de envio e de recebimento de cada mensagem, além de contar quantas mensagens foram recebidas. Neste experimento apenas os tamanhos das mensagens foram variados.

Os dados das mensagens recebidas foram armazenados em um arquivo CSV local no próprio dispositivo móvel e também foram enviados para o servidor principal, que armazenou os dados em um banco de dados SQL.

Os dados salvos no banco de dados também foram gerenciados pela aplicação DBeaver, que permite fazer a transição de dados de arquivos em SQL para CSV e vice-versa. Com essas ferramentas foi possível automatizar o processamento dos dados para análise.

Com os dados extraídos foi possível calcular as médias e os desvios padrão de cada uma das métricas de desempenho que foram analisadas, apresentadas na Seção 4.3.

4.2 Ambiente de execução e configurações

Os experimentos apresentados neste trabalho utilizaram os seguintes elementos:

- Dispositivo Android: Xiaomi Poco X3 Pro 256GB (Android 12);
- Servidor de mensagens: Raspberry Pi 4 4GB;
- Servidor principal: hospedado na Heroku;
- Banco de dados: hospedado na Heroku;

As configurações empregadas em cada experimento estão disponíveis no Quadro 1. No experimento para avaliar o consumo de energia foi variado o tempo entre mensagens. Nos demais experimentos, variou-se o tamanho das mensagens.

Quadro 1 – Configurações dos Experimentos

Configurações	Energia	Memória	Processamento	Rede
Tamanho da mensagem (caracteres)	1000	200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750 800, 850, 900, 950 e 1000	200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750 800, 850, 900, 950 e 1000	200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750 800, 850, 900, 950 e 1000
Tempo entre mensagens (ms)	100, 150, 200, 250 300, 350, 400, 450 e 500	250	250	250
Quantidade de mensagens (ms)	100	100	100	100
Repetições	30	30	30	30
Automação	desligada	desligada	desligada	ligada
Leitura dos dados	UNI-T UT658DUAL	Flutter DevTool	Flutter DevTool	arquivo CSV

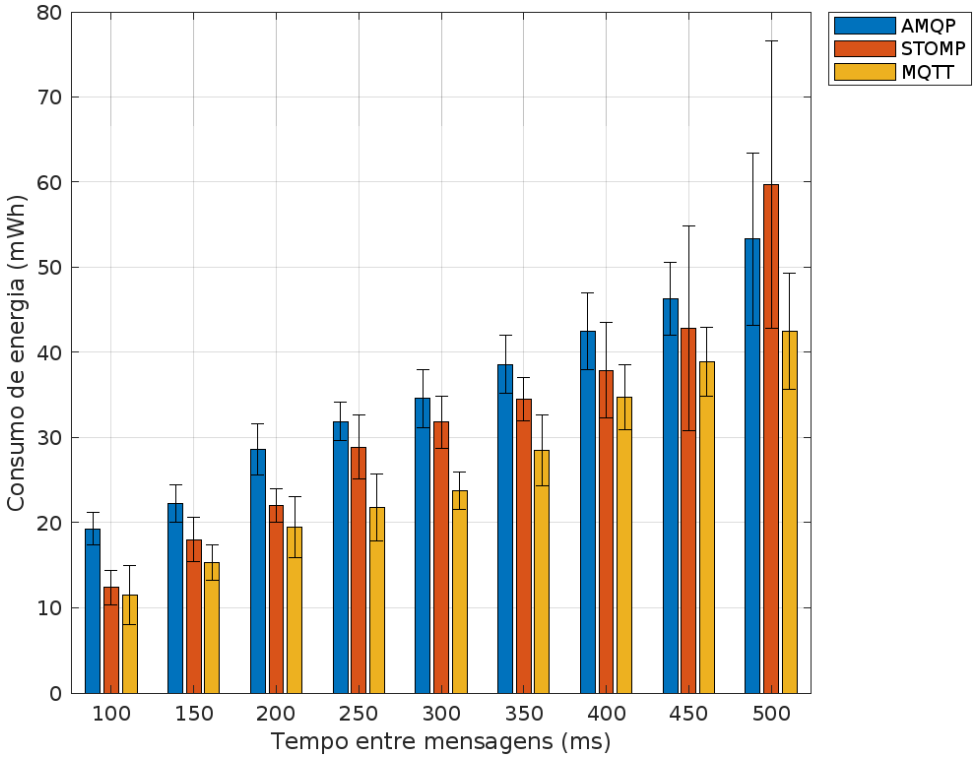
4.3 Resultados

Nesta seção são apresentados os resultados dos experimentos executados e suas respectivas análises. Os gráficos apresentados foram obtidos a partir dos dados das tabelas disponíveis no Apêndice A.

4.3.1 Consumo de energia

O Gráfico 1 mostra o resultado das medições do consumo de energia, a partir de um intervalo entre mensagens de 100 ms até 500 ms, variando 50 ms por série.

Gráfico 1 – Consumo de energia variando o intervalo entre mensagens



Fonte: Autoria própria.

Um intervalo mais restrito foi adotado neste experimento. Isto foi feito porque abaixo de 100 ms o sistema de chamadas assíncronas do Flutter começou a gerar erros para o MQTT. Já acima de 500 ms houve muita interferência do sistema operacional, que estava sempre realizando múltiplas tarefas para o funcionamento do dispositivo móvel.

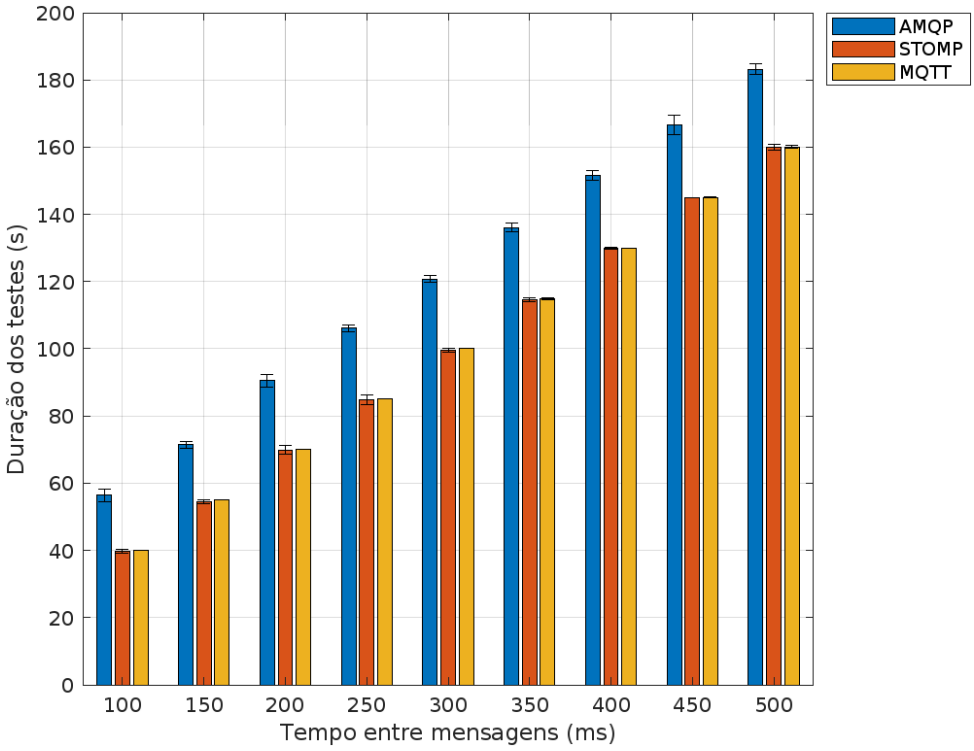
O efeito da interferência do sistema operacional pode ser notado nas medições para o intervalo de 500 ms, onde o desvio padrão, que aumentou a cada série, é o mais alto de todas as medições. Este efeito pode ser causado pela prioridade que o processador dá para cada tarefa que está tratando, que no caso da aplicação criada neste trabalho pode ter sido reduzida a cada nova série. Isto pode ser causado pela frequência cada vez menor de envio e recebimento de mensagens a cada nova série, o que faz diminuir a prioridade dos processos envolvidos no processamento e, consequentemente, o sistema operacional acaba realizando outras tarefas. Isto ocorre porque o Flutter é executado em um processo próprio e fica sujeito à preempção de tarefas do sistema operacional, que passa a priorizar processos de monitoramento do hardware embarcado.

Considerando apenas os valores médios para cada série, foi possível observar que o MQTT apresentou o menor consumo de energia entre todos os protocolos analisados. Quanto

aos outros dois protocolos observam-se dois cenários: abaixo de 500 ms, o STOMP foi o segundo melhor protocolo em termos de consumo de energia e o AMQP foi o pior entre os três, com o maior consumo de energia. Já no experimento com intervalo de 500 ms o comportamento observado foi oposto, sendo o consumo de energia do AMQP menor que o do STOMP. Porém, é importante ressaltar que, em alguns casos, os intervalos de confiança apresentados no gráfico se sobrepõem, indicando que os valores são estatisticamente iguais.

O Gráfico 2 mostra a duração dos experimentos de cada uma das séries de dados presentes no Gráfico 1, que consiste no tempo total para completar a transmissão de cada mensagem enviada.

Gráfico 2 – Duração média dos experimentos variando o intervalo entre mensagens



Fonte: Autoria própria.

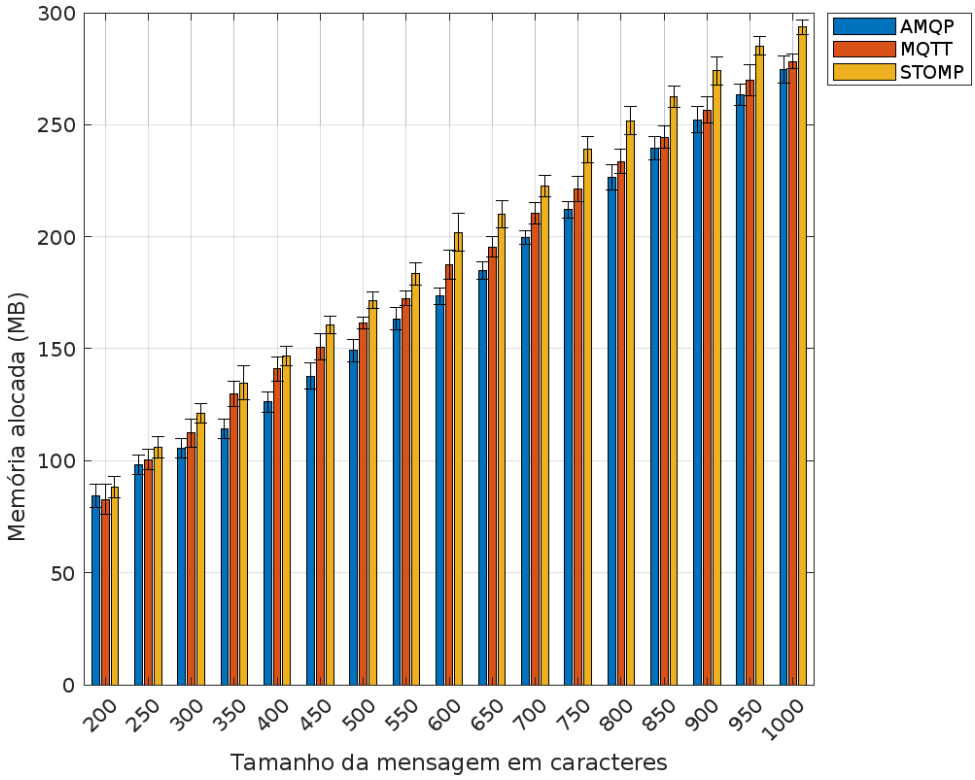
O Gráfico 2 mostra que o STOMP e MQTT ficaram muito próximos com relação ao tempo total para completar os experimentos, enquanto que o AMQP teve um desempenho pior.

4.3.2 Consumo de memória

O Gráfico 3 mostra os resultados dos experimentos de consumo de memória para os três protocolos, onde os tamanhos das mensagens foram variados de 200 caracteres até 1000 caracteres, com um passo de 50 caracteres entre cada série.

Na série de 200 caracteres, o MQTT apresentou um menor valor médio de consumo de memória, enquanto o AMQP obteve um resultado muito próximo. Já na série seguinte, de 250 caracteres, o AMQP se torna o protocolo com menor valor médio de consumo de memória, o

Gráfico 3 – Consumo de memória em função do tamanho das mensagens



Fonte: Autoria própria.

que resultou em uma inversão de posições com o MQTT. Este padrão se repetiu em todas as demais séries. O STOMP, por sua vez, foi o protocolo com maior consumo médio de memória em todos os cenários analisados.

Nas séries de 200, 250 e 300 caracteres os três protocolos apresentam interseções entre os valores de desvio padrão. Isto indica que, apesar de existirem diferenças entre os valores médios, os três protocolos não são estatisticamente diferentes.

O protocolo AMQP, entre o intervalo de 350 a 750 caracteres, não apresenta nenhuma interseção entre seu desvio padrão e o dos demais protocolos. Isto indica que os resultados são estatisticamente diferentes, sendo que o AMQP apresenta o menor valor para alocação de memória.

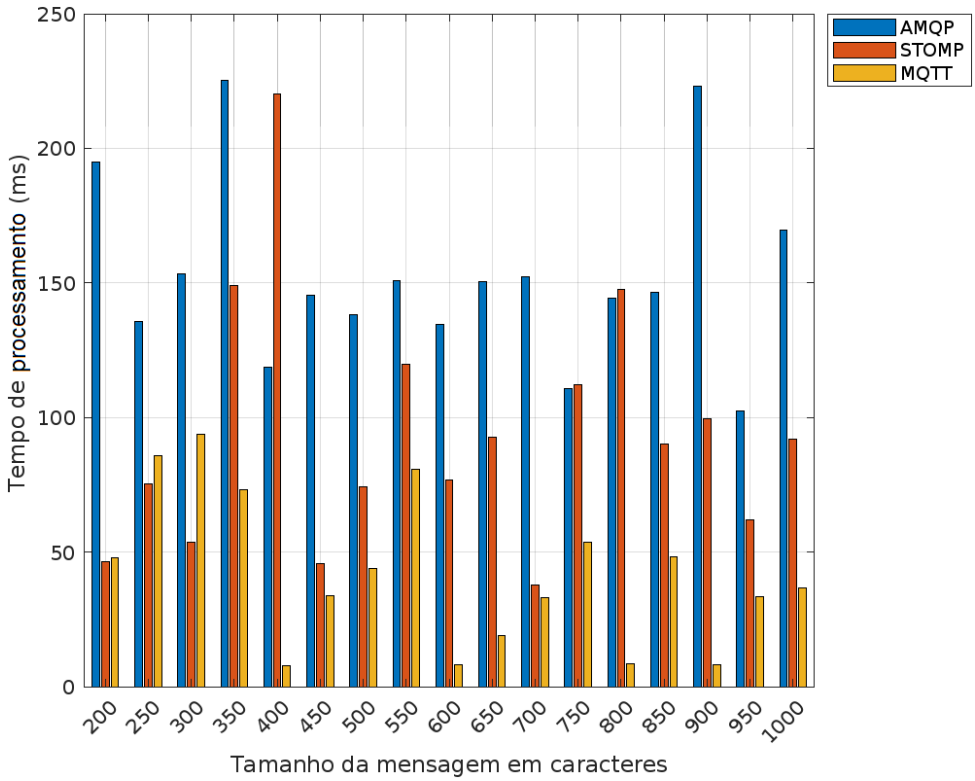
Já nas séries acima dos 750 caracteres existe uma intersecção entre os valores de desvio padrão do protocolo AMQP com o MQTT, o que indica que neste intervalo esses dois protocolos são estatisticamente iguais.

A partir da análise dos dados feita anteriormente, a conclusão é que o protocolo AMQP obteve os melhores resultados, de forma geral, para os experimentos de consumo de memória.

4.3.3 Uso de processamento

O Gráfico 4 mostra os resultados dos experimentos de uso de processador para os três protocolos, onde os tamanhos das mensagens foram variados de 200 caracteres até 1000 caracteres, com um passo de 50 caracteres entre cada série.

Gráfico 4 – Tempo de processamento em função do tamanho das mensagens



Fonte: Autoria própria.

Apesar de neste experimento, para cada série, terem sido realizadas várias repetições, todas apresentaram os mesmos resultados. Portanto, não existem valores de desvio padrão para nenhuma das séries para os três protocolos.

Entre as séries de 200 a 300 caracteres, o protocolo MQTT apresentou um maior uso de processador que o STOMP, que obteve os menores valores dos três protocolos. Entretanto, a partir da série de 350 caracteres, o protocolo MQTT obteve os menores valores de uso de processador em comparação com os outros dois protocolos.

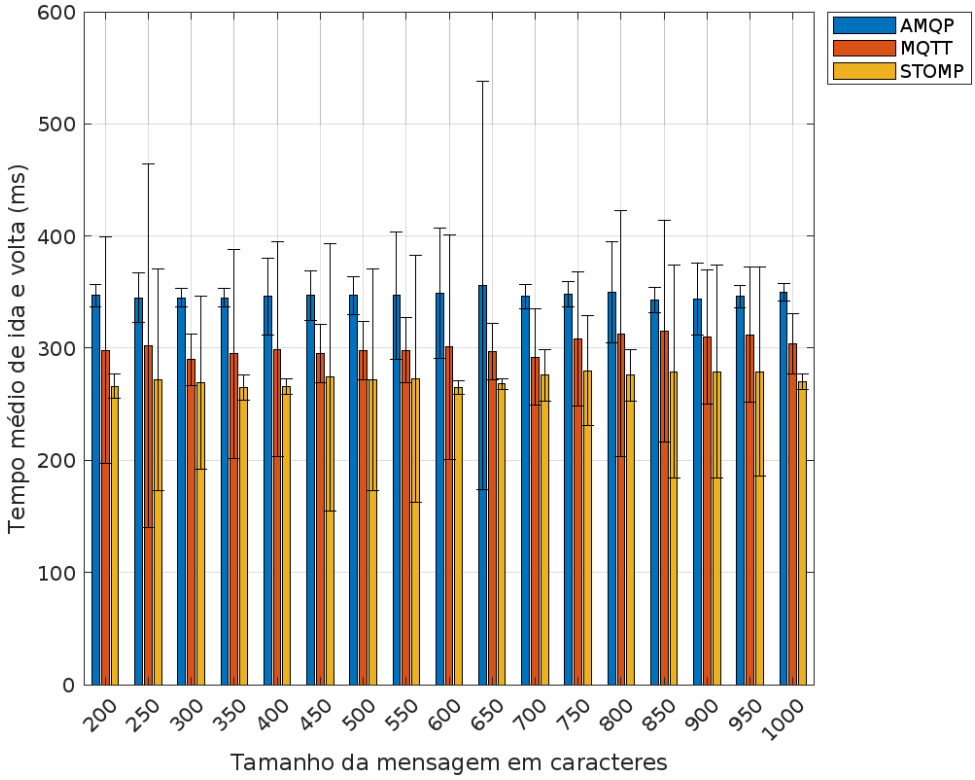
Portanto, a partir da análise dos dados feita anteriormente, a conclusão é que o protocolo MQTT obteve os melhores resultados, de forma geral, para os experimentos de uso de processamento.

Vale pontuar que os Gráficos 3 e 4 indicam que o protocolo AMQP usa maior processamento para compensar o menor uso de memória. O MQTT faz o contrário, pois utiliza mais memória para compensar processamento.

4.3.4 Tempo de ida e volta das mensagens

O Gráfico 5 mostra os resultados dos experimentos para o tempo de ida e volta de mensagens considerando os três protocolos, onde os tamanhos das mensagens foram variados em 50 caracteres, iniciando em 200 caracteres até 1000 caracteres.

Gráfico 5 – Tempo de ida e volta de uma mensagem em função do tamanho da mensagem



Fonte: Autoria própria.

Os tempos médios de ida e volta de mensagem de cada protocolo se mantiveram próximos durante todas as séries, com pequenas variações entre eles. A alteração do tamanho das mensagens afetou pouco o tempo médio de ida e volta das mensagens para os três protocolos. Considerando apenas os resultados dos experimentos para os tempos médios, conclui-se que o protocolo que obteve os melhores resultados foi o STOMP, seguido pelo MQTT e, por fim, pelo AMQP.

É possível perceber que na maioria das séries quanto maior o desvio padrão, maior foi o tempo médio de ida e volta das mensagens.

No experimento com mensagens de 650 caracteres utilizando o AMQP, pode-se perceber a existência de valores altos de desvio padrão. Nesse caso, o tempo de transmissão da maioria das mensagens estava próximo da média presente, porém existiam algumas amostras em que o tempo total de transmissão foi na ordem de segundos. Considerando a aleatoriedade com que isto ocorria e que o servidor de mensagens foi executado em um computador exclusivo para ele, em uma rede local, este comportamento indica que essas variações são decorrentes de interferências causadas pelo dispositivo móvel.

É importante, porém, notar que em várias séries o tempo médio de cada protocolo está dentro do intervalo de confiança dos outros dois protocolos, o que indica que os valores não são estatisticamente diferentes.

4.3.5 Perda de mensagens

Durante os experimentos, considerando os três protocolos analisados, não houve perda de mensagens. Todas as mensagens foram entregues corretamente e na mesma ordem em que foram enviadas.

5 CONCLUSÃO

Este trabalho consistiu em uma análise de desempenho dos protocolos STOMP, MQTT e AMQP em um ambiente de computação em névoa. O intuito da análise foi observar o desempenho de cada um dos protocolos quanto às métricas de consumo de energia, memória, processamento, tempo de ida e volta e perda de mensagens em um dispositivo móvel com sistema operacional Android 12.

A área de computação móvel com uso de computação em névoa cresce a cada dia, onde o uso dos protocolos analisados neste trabalho é muito alto, porém, existem poucos estudos publicados sobre essa área até o momento. Este trabalho pode ajudar a comunidade a criar um entendimento melhor sobre esses protocolos e auxiliar pessoas na escolha deles para a implementação em projetos.

Diante dos resultados obtidos no presente trabalho, alguns apontamentos sobre cada protocolo podem ser feitos. Cada protocolo apresentou benefícios em determinadas áreas da análise, de modo que a indicação de um protocolo depende muito das limitações do sistema no qual o protocolo for implementado. Sendo assim, é necessário avaliar os requisitos de projeto para determinar qual é a melhor escolha para uma determinada aplicação. Quando existirem limitações quanto ao uso de processamento, o protocolo mais recomendado é o MQTT. Caso o consumo de memória do dispositivo seja limitado, então o protocolo mais recomendado é o AMQP. Caso o problema seja consumo de energia, os protocolos MQTT e STOMP são os mais recomendados. Como em todos os experimentos o MQTT obteve os melhores resultados ou foi o protocolo com o segundo melhor desempenho, ele seria o protocolo mais recomendado de modo geral.

Vale destacar que existem várias opções de configuração e transmissão de mensagens para cada um dos protocolos, o que também afeta na escolha deles. Dependendo do tamanho e complexidade do projeto, como o uso de dispositivos heterogêneos, é possível implementar mais de um protocolo.

Como trabalhos futuros, seria interessante fazer uma avaliação dos mesmos protocolos (STOMP, MQTT e AMQP), porém com o uso da programação nativa em Android, para que assim possam ser realizadas comparações com os dados obtidos neste trabalho. Estas novas análises poderiam permitir avaliar eventuais interferências que possam ter sido causadas devido ao uso do *Flutter*, já que com a programação nativa talvez seja possível ter mais controle sobre o sistema operacional. Além disso, em um trabalho futuro pode-se analisar o desempenho dos protocolos considerando diferentes configurações de transmissão de mensagens e utilizando diferentes tipos de dispositivos móveis.

Todo o código de programação utilizado na implementação deste projeto pode ser encontrado no GitHub: https://github.com/beltraoluis/TCC-Mobile-EngComp2021.

REFERÊNCIAS

AMANATULLAH, Y. *et al.* Toward cloud computing reference architecture: Cloud service management perspective. **International Conference on ICT for Smart Society**, IEEE, 2013.

AMAZON. **What Is Cloud Computing?** 2018. Disponível em: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/what-is-cloud-computing.html. Acesso em: 17 Abr. 2022.

AMERICA, D. S. C. of. **Top 6 Programming Languages for Data Science in 2021**. 2021. Disponível em: https://www.dasca.org/world-of-big-data/article/top-6-programming-languages-for-data-science-in-2021. Acesso em: 05 Mai. 2022.

ANDROID, D. **Mudanças de comportamento: todos os apps**. 2019. Disponível em: https://developer.android.com/about/versions/10/behavior-changes-all. Acesso em: 22 Ago. 2022.

ARDC, A. R. D. C. **Standardised communications protocols**. 2020. Disponível em: https://ardc.edu.au/resources/standardised-communications-protocols/. Acesso em: 17 Jul. 2022.

AZEVEDO, M. T. **Transformação digital na indústria: indústria 4.0 e a rede de água inteligente no Brasil**. 2017. Tese (Doutorado) — Universidade de São Paulo, 2017.

BEZERRA, W. R.; WESTPHALL, C. B. Avaliação de desempenho de protocolos de mensagens com arquitetura publish/subscribe no ambiente de computação em nevoeiro: um estudo sobre desempenho do mqtt, amqp e stomp. **2020: Anais do XI Workshop de Pesquisa Experimental da Internet do Futuro**, Sociedade Brasileira de Computação, 2020.

BORMANN, C. **CoAP - RFC 7252 Constrained Application Protocol**. 2016. Disponível em: https://coap.technology/. Acesso em: 17 Jul. 2022.

CISCO. Fog computing and the internet of things: Extend the cloud to where the things are. **White Paper - Cisco**, Cisco, 2015.

CONSORTIUM, O. **Introduction and Overview at W3C Open Day**. 2017. Disponível em: https://www.w3.org/2017/05/wot-f2f/slides/OpenFog-Overview-W3C-Open-Day-in-May-2017.pdf. Acesso em: 18 Abr. 2022.

COPE, S. **Beginners Guide To The MQTT Protocol**. 2018. Disponível em: http://www.steves-internet-guide.com/mqtt/. Acesso em: 05 Mai. 2022.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems**: Concepts and design. [*S.l.*]: Addison Wesley Longman, 2011. 242 - 253 p.

COUTINHO, A. A. T. R.; CARNEIRO, E. O.; GREVE, F. Computação em névoa: Conceitos, aplicações e desafios. **XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, v. 6, p. 266–315, 2016.

COUTINHO, G. L. A era dos smartphones: um estudo exploratório sobre o uso dos smartphones no brasil. **Universidade de Brasília**, UNB, 2014. Disponível em: hhttps://bdm.unb.br/handle/10483/9405. Acesso em: 05 mai. 2022.

CROSSBAR.IO. **The Web Application Messaging Protocol**. 2022. Disponível em: https://wamp-proto.org/. Acesso em: 02 Set. 2022.

DART, E. **Dart documentation**. 2022. Disponível em: https://dart.dev/guides. Acesso em: 05 Mai. 2022.

DART_AMQP, C. do. **dart_amqp 0.2.1**. 2021. Disponível em: https://pub.dev/packages/dart_amqp. Acesso em: 05 Mai. 2022.

DDS, D. F. **Data Distribution Service**. 2022. Disponível em: https://www.dds-foundation.org/. Acesso em: 17 Jul. 2022.

FLASK. **User's Guide**. 2022. Disponível em: https://flask.palletsprojects.com/en/2.0.x/. Acesso em: 05 Mai. 2022.

FOUNDATION, O. **Unified Architecture**. 2022. Disponível em: https://opcfoundation.org/about/opc-technologies/opc-ua/. Acesso em: 21 Jul. 2022.

GOOGLE, F. T. **Flutter documentation**. 2022. Disponível em: https://docs.flutter.dev/. Acesso em: 05 Mai. 2022.

GROUP, T. P. G. D. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2022. Disponível em: https://www.postgresql.org/. Acesso em: 05 Mai. 2022.

HEROKU. **Learn about building, deploying, and managing your apps on Heroku.** 2022. Disponível em: https://devcenter.heroku.com/. Acesso em: 05 Mai. 2022.

HTTP.DART, C. do. **http 0.13.4**. 2021. Disponível em: https://pub.dev/packages/http. Acesso em: 05 Mai. 2022.

INC, D. **Docker overview**. 2022. Disponível em: https://docs.docker.com/get-started/overview/. Acesso em: 05 Mai. 2022.

IRONS-MCLEAN, R.; SABELLA, A.; YANNUZZI, M. Iot and security standards and best practices. **Cisco Press**, Cisco, 2019.

ISO, I. O. for S. **ABOUT US**. 2022. Disponível em: https://www.iso.org/about-us.html. Acesso em: 18 Jul. 2022.

JOHNSEN, F. T. Using publish/subscribe for short-lived iot data. **2018 Federated Conference on Computer Science and Information Systems (FedCSIS)**, IEEE, 2018.

KALE, V. **Guide to Cloud Computing for Business and Technology Managers**: From distributed computing to cloudware applications. [*S.l.*]: Chapman and Hall/CRC, 2014. 117 - 120 p.

KITAMURA, C. **O Que É Request E Response?** 2021. Disponível em: https://celsokitamura.com.br/o-que-e-request-e-response/. Acesso em: 18 Jul. 2022.

LUZURIAGA, J. E. *et al.* A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. **2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)**, IEEE, 2015.

MAGNONI, L. Modern messaging for distributed sytems. **Journal of Physics: Conference Series**, Journal of Physics, 2014.

MATHWORKS. **MATLAB - Overview**. 2022. Disponível em: https://www.mathworks.com/products/matlab.html. Acesso em: 05 Mai. 2022.

MIMOUNI, S. E.; BOUHDADI, M. Formal modeling of the simple text oriented messaging protocol using event-b method. **2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)**, IEEE, 2015.

MOZILLA. **WebSockets**. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/API/WebSockets_API. Acesso em: 21 Jul. 2022.

MQTT_CLIENT, C. do. **mqtt_client 9.6.7**. 2022. Disponível em: https://pub.dev/packages/mqtt_client. Acesso em: 05 Mai. 2022.

MQTT.ORG. **MQTT: The Standard for IoT Messaging**. 2022. Disponível em: https://mqtt.org/. Acesso em: 17 Jul. 2022.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. **2017 IEEE International Systems Engineering Symposium (ISSE)**, IEEE, 2017.

NEST, G. **OpenWeave**. 2022. Disponível em: https://openweave.io/. Acesso em: 30 Ago. 2022.

NIST. **The NIST Definition of Cloud Computing**. 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Acesso em: 18 Abr. 2022.

OASIS. **MQTT Version 3.1.1**. 2014. Disponível em: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. Acesso em: 05 Mai. 2022.

OASIS. **About Us**. 2022. Disponível em: https://www.oasis-open.org/org/. Acesso em: 17 Jul. 2022.

OASIS. **AMPQ - Advanced Message Queuing Protocol**. 2022. Disponível em: https://www.amqp.org/. Acesso em: 17 Jul. 2022.

O'RIORDAN, M. **Everything You Need To Know About Publish/Subscribe**. 2020. Disponível em: https://ably.com/topic/pub-sub. Acesso em: 30 Ago. 2022.

O'RIORDAN, M. **Publish-Subscribe: Introduction to Scalable Messaging**. 2020. Disponível em: https://thenewstack.io/publish-subscribe-introduction-to-scalable-messaging/. Acesso em: 30 Ago. 2022.

PATEL, K. K.; PATEL, S. M. Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application future challenges. **Faculty of Technology and Engineering-MSU**, 2016.

PSYCOPG2, E. **Project description**. 2022. Disponível em: https://pypi.org/project/psycopg2/. Acesso em: 05 Mai. 2022.

RABBITMQ. **AMQP 0-9-1 Model Explained**. 2011. Disponível em: https://www.rabbitmq.com/tutorials/amqp-concepts.html. Acesso em: 05 Mai. 2022.

RABBITMQ. **RabbitMQ is the most widely deployed open source message broker.** 2022. Disponível em: https://www.rabbitmq.com/. Acesso em: 05 Mai. 2022.

RABBITMQ. **Server Operator Documentation**. 2022. Disponível em: https://www.rabbitmq.com/admin-guide.html. Acesso em: 05 Mai. 2022.

ROSS, A. B. Uso da computação em névoa para coleta e análise de dados em cidades inteligentes. trabalho de conclusão de curso (graduação). **Universidade Tecnológica Federal do Paraná**, UTFPR, 2019.

SARKAR, S.; CHATTERJEE, S.; MISRA, S. Assessment of the suitability of fog computing in the context of internet of things. **IEEE Transactions on Cloud Computing**, IEEE, 2015.

SATYANARAYANAN, M. Fundamental challenges in mobile computing. **Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing**, Carnegie Mellon University, 1996.

SHUIB, L.; SHAMSHIRBAND, S.; ISMAIL, M. H. A review of mobile pervasive learning: Applications and issues. **Computers in Human Behavior**, Elsevier, 2015.

SILVA, D. A. J. Desenvolvimento e construção de processadores: Uma breve história da micro a nanotecnologia. **Universidade Federal do Maranhão**, 2017.

SILVEIRA, P. **O que é SQL?** 2019. Disponível em: https://www.alura.com.br/artigos/o-que-e-sql. Acesso em: 05 Mai. 2022.

SOUZA, F. **MQTT-SN: MQTT para rede de sensores**. 2018. Disponível em: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn. Acesso em: 18 Jul. 2022.

SPECWORKS, O. **Lightweight M2M (LWM2M)**. 2022. Disponível em: https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/. Acesso em: 21 Jul. 2022.

STOMP. **STOMP Protocol Specification, Version 1.2**. 2013. Disponível em: https://stomp.github.io/stomp-specification-1.2.html. Acesso em: 05 Mai. 2022.

STOMP.DART, C. do. **stomp 0.8.0**. 2019. Disponível em: https://pub.dev/packages/stomp. Acesso em: 05 Mai. 2022.

TEAM, D. **About DBeaver**. 2021. Disponível em: https://github.com/dbeaver/dbeaver/wiki. Acesso em: 05 Mai. 2022.

UY, N. Q.; NAM, V. H. A comparison of amqp and mqtt protocols for internet of things. **2019 6th NAFOSTED Conference on Information and Computer Science (NICS)**, IEEE, 2019.

WALLACE, S.; CLARK, M.; WHITE, J. 'it's on my iphone': attitudes to the use of mobile computing devices in medical education, a mixed-methods study. **BMJ Open 2012**, BMJ Open, 2012.

WYTREBOWICZ, J.; CABAJ, K.; KRAWIEC, J. Messaging protocols for iot systems — a pragmatic comparison. **Sensors**, MDPI, 2021.

XSF, X. S. F. **The universal messaging standard**. 2022. Disponível em: https://xmpp.org/. Acesso em: 17 Jul. 2022.

APÊNDICE A – Tabelas de dados experimentais

Tabela 1 – Potência consumida nos experimentos de energia

protocolo \ série	100	150	200	250	300
AMQP (mWh)	$19{,}3 \pm 1{,}9$	$22{,}3 \pm 2{,}2$	$28{,}6 \pm 3$	$31{,}9 \pm 2{,}3$	$34{,}6 \pm 3{,}4$
STOMP (mWh)	$12{,}4 \pm 2{,}0$	$18 \pm 2{,}6$	$22 \pm 2{,}0$	$28{,}9 \pm 3{,}8$	$31{,}8 \pm 3{,}1$
MQTT (mWh)	$11{,}5 \pm 3{,}5$	$15{,}3 \pm 2{,}1$	$19{,}5 \pm 3{,}6$	$21{,}8 \pm 3{,}9$	$23{,}8 \pm 2{,}2$
protocolo \ série	350	400	450	500	
AMQP (mWh)	$38{,}6 \pm 3{,}4$	$42{,}5 \pm 4{,}5$	$46{,}3 \pm 4{,}3$	$53{,}3 \pm 10{,}1$	
STOMP (mWh)	$34{,}5 \pm 2{,}5$	$37{,}9 \pm 5{,}6$	$42{,}8 \pm 12$	$59{,}7 \pm 16{,}9$	
MQTT (mWh)	$28{,}5 \pm 4{,}2$	$34{,}7 \pm 3{,}8$	$38{,}9 \pm 4{,}0$	$42{,}5 \pm 6{,}8$	

Os dados da Tabela 1 foram empregados na geração do Gráfico 1.

Tabela 2 – Duração dos experimentos de energia

protocolo \ série	100	150	200	250	300
AMQP (s)	$56{,}4 \pm 1{,}9$	$71{,}4 \pm 1$	$90{,}5 \pm 1{,}9$	$106{,}1 \pm 1{,}1$	$120{,}8 \pm 1$
STOMP (s)	$39{,}8 \pm 0{,}5$	$54{,}5 \pm 0{,}5$	$69{,}9 \pm 1{,}3$	$84{,}8 \pm 1{,}4$	$99{,}5 \pm 0{,}5$
MQTT (s)	40	55	70	85	100
protocolo \ série	350	400	450	500	
AMQP (s)	$136{,}1 \pm 1{,}4$	$151{,}5 \pm 1{,}5$	$166{,}6 \pm 3$	$183{,}2 \pm 1{,}6$	
STOMP (s)	$114{,}6 \pm 0{,}5$	$129{,}9 \pm 0{,}3$	145	$159{,}9 \pm 0{,}9$	
MQTT (s)	$115 \pm 0{,}3$	130	$145 \pm 0{,}2$	$160{,}1 \pm 0{,}4$	

Os dados da Tabela 2 foram empregados na geração do Gráfico 2.

Tabela 3 – Consumo de memória

protocolo \ série	200	250	300	350	400
AMQP (MB)	$84{,}6 \pm 5{,}2$	$98{,}2 \pm 4{,}3$	$105{,}7 \pm 4{,}3$	$114{,}3 \pm 4{,}2$	$126{,}3 \pm 4{,}5$
STOMP (MB)	$88{,}4 \pm 4{,}9$	$106{,}0 \pm 4{,}7$	$121{,}1 \pm 4{,}4$	$134{,}8 \pm 7{,}6$	$146{,}7 \pm 4{,}4$
MQTT (MB)	$82{,}9 \pm 6{,}7$	$100{,}6 \pm 4{,}6$	$112{,}4 \pm 6{,}3$	$129{,}8 \pm 5{,}6$	$141{,}0 \pm 5{,}4$
protocolo \ série	450	500	550	600	650
AMQP (MB)	$137{,}8 \pm 5{,}8$	$149{,}3 \pm 5{,}0$	$163{,}4 \pm 5{,}0$	$173{,}7 \pm 3{,}7$	$184{,}9 \pm 3{,}8$
STOMP (MB)	$160{,}8 \pm 4{,}0$	$171{,}7 \pm 3{,}7$	$183{,}5 \pm 5{,}0$	$202{,}0 \pm 8{,}6$	$210{,}0 \pm 6{,}0$
MQTT (MB)	$150{,}9 \pm 5{,}8$	$161{,}5 \pm 2{,}6$	$172{,}6 \pm 3{,}1$	$187{,}3 \pm 6{,}5$	$195{,}5 \pm 4{,}5$
protocolo \ série	700	750	800	850	900
AMQP (MB)	$199{,}9 \pm 3{,}0$	$212{,}0 \pm 3{,}6$	$226{,}6 \pm 5{,}7$	$239{,}6 \pm 5{,}3$	$252{,}4 \pm 5{,}8$
STOMP (MB)	$222{,}6 \pm 4{,}7$	$239{,}1 \pm 5{,}8$	$251{,}9 \pm 6{,}3$	$262{,}6 \pm 4{,}9$	$274{,}1 \pm 6{,}1$
MQTT (MB)	$210{,}4 \pm 4{,}8$	$221{,}4 \pm 5{,}5$	$233{,}7 \pm 5{,}4$	$244{,}4 \pm 4{,}9$	$256{,}7 \pm 5{,}7$
protocolo \ série	950	1000			
AMQP (MB)	$263{,}5 \pm 4{,}7$	$274{,}8 \pm 6{,}1$			
STOMP (MB)	$285{,}3 \pm 4{,}0$	$293{,}6 \pm 3{,}4$			
MQTT (MB)	$270{,}0 \pm 6{,}9$	$278{,}3 \pm 3{,}4$			

Os dados da Tabela 3 foram empregados na geração do Gráfico 3.

Tabela 4 – Tempo de processamento

protocolo \ série	200	250	300	350	400
AMQP (ms)	195,03	135,61	153,18	225,39	118,60
STOMP (ms)	46,39	75,26	53,73	149,09	220,21
MQTT (ms)	47,81	85,81	93,90	73,00	7,88
protocolo \ série	450	500	550	600	650
AMQP (ms)	145,36	138,11	150,69	134,41	150,51
STOMP (ms)	45,83	74,08	119,9	76,89	92,60
MQTT (ms)	33,81	43,95	80,85	7,96	18,89
protocolo \ série	700	750	800	850	900
AMQP (ms)	152,45	110,69	144,25	146,57	223,11
STOMP (ms)	37,86	112,14	147,57	90,01	99,66
MQTT (ms)	33,02	53,79	8,63	48,07	8,27
protocolo \ série	950	1000			
AMQP (ms)	102,57	169,79			
STOMP (ms)	61,98	91,86			
MQTT (ms)	33,34	36,60			

Os dados da Tabela 4 foram empregados na geração do Gráfico 4.

Tabela 5 – Tempo de ida e volta

protocolo \ série	200	250	300	350	400
AMQP (ms)	347 ± 11	345 ± 22	345 ± 8	345 ± 8	346 ± 34
STOMP (ms)	266 ± 11	272 ± 99	269 ± 77	265 ± 11	266 ± 7
MQTT (ms)	298 ± 101	302 ± 162	290 ± 23	295 ± 93	299 ± 96
protocolo \ série	450	500	550	600	650
AMQP (ms)	347 ± 22	347 ± 17	347 ± 57	349 ± 58	356 ± 182
STOMP (ms)	274 ± 119	272 ± 99	273 ± 110	265 ± 6	268 ± 5
MQTT (ms)	295 ± 26	298 ± 26	298 ± 29	301 ± 100	297 ± 25
protocolo \ série	700	750	800	850	900
AMQP (ms)	346 ± 11	348 ± 11	350 ± 45	343 ± 11	344 ± 32
STOMP (ms)	276 ± 23	280 ± 49	276 ± 23	279 ± 95	279 ± 95
MQTT (ms)	292 ± 43	308 ± 60	313 ± 110	315 ± 99	310 ± 60
protocolo \ série	950	1000			
AMQP (ms)	346 ± 10	350 ± 8			
STOMP (ms)	279 ± 93	270 ± 7			
MQTT (ms)	312 ± 60	304 ± 27			

Os dados da Tabela 5 foram empregados na geração do Gráfico 5.