
**IGOR OTÁVIO CAETANO DINIZ**

# UM ESTUDO COMPARATIVO DE PROTOCOLOS DE APLICAÇÃO IOT

**LAVRAS – MG**

**2022**

**IGOR OTÁVIO CAETANO DINIZ**

# UM ESTUDO COMPARATIVO DE PROTOCOLOS DE APLICAÇÃO IOT

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Neumar Costa Malheiros
Orientador

**LAVRAS – MG**

**2022**

**IGOR OTÁVIO CAETANO DINIZ**

# UM ESTUDO COMPARATIVO DE PROTOCOLOS DE APLICAÇÃO IOT

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 09 de Setembro de 2022.

Prof. Luiz Henrique Andrade Correia UFLA
Prof. Hermes Pimenta de Moraes Júnior UFLA

Prof. Dr. Neumar Costa Malheiros
Orientador

**LAVRAS – MG**
**2022**

*Dedico este trabalho a todos futuros arquitetos de IoT na esperança que este possa ajudá-los futuramente.*

**AGRADECIMENTOS**

Primeiramente, agradeço a mim mesmo por não ter desistido mesmo diante de tantas adversidades. A minha mãe e meus avós por me incentivarem e me apoiar nessa caminhada, mesmo quando tudo parecia perdido. Aos meus amigos que sempre me motivaram e me convenceram que eu era capaz. E por último, mas não menos importante, aos professores, por todos os conselhos, pela ajuda e pela paciência com a qual guiaram o meu aprendizado.

*Um espírito nobre engrandece o menor dos homens.*
*(Theodore Roosevelt)*

# RESUMO

Com o crescimento da demanda por soluções de Internet das Coisas (IoT), desenvolvedores precisam escolher de forma criteriosa qual protocolo utilizar em sua aplicação. Essa escolha pode acabar se tornando complexa à medida em que existem vários protocolos, cada um com suas especificidades e diferenças de funcionamento. Este trabalho tem como intuito introduzir conceitos de IoT e descrever alguns dos principais protocolos, desta forma, auxiliando arquitetos de soluções a escolherem de maneira coerente um protocolo que melhor lhes atenderá em suas demandas. Em particular, serão abordados três protocolos na camada de aplicação: MQTT e COAP, AMQP. Por fim, será apresentada uma análise comparativa entre os protocolos, considerando suas principais funcionalidades e características.

**Palavras-chave:** Internet das Coisas (IoT). Camada de Aplicação. MQTT. COAP. AMQP.

# ABSTRACT

With the growing demand for Internet of Things (IoT) solutions, developers need to carefully choose which protocol to use in their application. This choice can end up becoming complex as there are several protocols, each with its specificities and differences in operation. This work aims to introduce IoT concepts and describe some of the main protocols, thus helping solution architects to coherently choose a protocol that will best meet their demands. In particular, three protocols will be addressed at the application layer level: MQTT and COAP, AMQP. Finally, a comparative analysis between the protocols will be presented, considering their main functionalities and characteristics.



**Keywords:** Internet of Things (IoT). Application Layer. MQTT. COAP. AMQP.

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

Com o avanço constante da tecnologia, o uso de aplicações embarcadas para criar sistemas que possam se comunicar com outros dispositivos é algo cada vez mais requisitado por diferentes indústrias e serviços. Geralmente, essas aplicações embarcadas atuam com vários fatores limitantes, entre os quais pode-se destacar: baixo poder de processamento e uma conexão instável de internet. Assim, surgiu o paradigma de Internet das Coisas (IoT – *Internet of Things*), que pode ser definido como uma rede de sensores incorporados a objetos físicos ou componentes de software capazes de serem identificados, realizar processamento de dados, e se comunicar com outros dispositivos através da Internet (ORACLE, 2022).

Com o passar do tempo, surgiram vários protocolos para IoT, cada um com um propósito de tratar alguma limitação dos dispositivos e dos sistemas de comunicação. Com a grande quantidade de protocolos, um arquiteto de soluções tem uma tarefa complexa para resolver quando precisa tomar uma decisão sobre qual é o melhor protocolo IoT para determinada aplicação. Isso ocorre, principalmente, por causa das grandes diferenças que podem surgir de um protocolo para outro em termos de funcionalidades e desempenho.

Este trabalho tem o intuito de realizar um estudo comparativo entre diversos protocolos de IoT a fim de auxiliar arquitetos de soluções e desenvolvedores de aplicações em sua tomada de decisão no momento de escolher um protocolo IoT na camada de aplicação. Neste estudo, será realizada uma comparação, qualitativa e quantitativa, entre três dos principais protocolos IoT na camada de aplicação, que são mais utilizados atualmente, sendo eles: *MQTT*, *CoAP* e *AMQP* (AL-MASRI et al., 2020).

Este texto está estruturado da seguinte forma. No Capítulo 2, serão introduzidos os conceitos básicos sobre IoT e também serão descritas as principais características dos protocolos abordados neste trabalho. No Capítulo 3, será realizada uma análise comparativa entre os protocolos. E, por último, no Capítulo 4, serão apresentadas as considerações finais.

# 2 REFERENCIAL TEÓRICO

Neste capítulo, serão abordados os conceitos básicos do paradigma de IoT e também serão apresentadas as características principais dos protocolos MQTT, CoAP e AMQP, que são protocolos na camada de aplicação para IoT.

## 2.1 Fundamentos de IoT

O surgimento de dispositivos inteligentes capazes de se comunicar com outros dispositivos via uma rede começou em 1982, quando pesquisadores da Universidade de Carnegie Mellon desenvolveram uma máquina de vender refrigerante, capaz de identificar e catalogar quando novas bebidas, adicionadas à máquina, ficassem geladas. Este foi o primeiro passo para o desenvolvimento de novos dispositivos capazes de realizar tarefas de comunicação com outros dispositivos de forma autônoma, sem nenhuma interferência humana no processo (SONI; MAKWANA, 2017).

Gokhale, Bhat e Bhat (2018) citam que em 1999 Kevin Ashton utilizou pela primeira vez o termo IoT (*Internet of Things*) para se referir a dispositivos que utilizavam redes sem fio para transmitir informações e instruções de uns para outros.

Apesar de hoje existir essa definição de IoT como uma comunicação entre dispositivos utilizando uma rede sem fio, quando Kevin Ashton propôs o termo, inicialmente ele havia pensando em uma outra definição, conforme Gokhale, Bhat e Bhat (2018) destacam:

> Kevin Ashton inicialmente propôs o conceito de IoT em 1999, e se referiu ao IoT como somente objetos identificavelmente conectados pela tecnologia de um identificador de radio-frequência (RFID – *radio-frequency identification*). Porém, a exata definição de IoT ainda está em processo de formação que é assunto para outras perspectivas. IoT foi definido genericamente como "Infraestrutura de network dinâmica global com capacidade de se autoconfigurar baseado em premissas e protocolos de comunicações"

Nos últimos anos, a quantidade de dispositivos IoT vem aumentando cada vez mais. Com o avanço das redes sem fio, diversos dispositivos agora podem ser utilizados para se conectar à Internet. Estes dispositivos por sua vez, acoplam uma grande quantidade de diversos tipos de sensores, que lhes permitem coletar uma grande quantidade de dados, processar, e informar para outros dispositivos ou até mesmo diretamente para o usuário, alguma informação que ele ainda não estava ciente.

É possível apontar, como exemplo, *smartwatches*, que atualmente contêm diversos tipos de sensores como GPS, sensor óptico de batimentos, acelerômetro, giroscópio, dentre outros.

Dentro do contexto de IoT, esses *smartwatches* são considerados*processadores*. Conforme explicado em (TAKIDDEEN; ZUALKERNAN, 2019), os nós processadores participam de quatro etapas em um ciclo de aplicação IoT. Essas etapas são:

- Aquisição de contexto, que é o momento em que são coletados os dados através dos sensores;
- Modelagem de contexto, cujo papel é agrupar os dados coletados colocados em um formato para serem processados;
- Contexto lógico, que envolve extrair os dados modelados e prepará-los para disseminação;
- Contexto de disseminação, na qual os dados processados são transmitidos para o usuário ou outros dispositivos.

### 2.1.1 Arquitetura de Referência

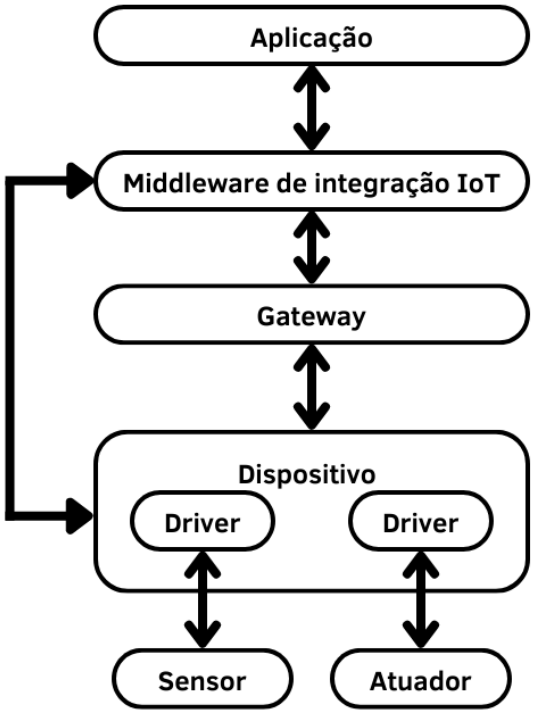
Quando se fala sobre IoT, é preciso entender a estrutura do sistema, quais são os elementos envolvidos e como eles interagem entre si. Neste contexto, Guth et al. (2016) nos apresentam um modelo de referência para a arquitetura de soluções de IoT, conforme ilustrado na Figura 2.1.

O modelo de referência proposto é genérico, de alto nível de abstração e pode sofrer variações de sistema para sistema. Pode-se notar que uma arquitetura IoT pode ser composta de: dispositivo com seus sensores e atuadores, *gateway* e um *middleware* de integração IoT.

O sensor é um componente de hardware usado para aferir dados de um ambiente e traduzir essas informações para sinais elétricos. Por exemplo, medir a temperatura de uma sala. Um sensor pode ser configurando utilizando algum software, porém, este sensor não pode, ele mesmo, executar o software. Este processo de configuração deve ser feito a parte. O sensor tem como única função coletar as informações.

O Atuador também é um componente de hardware. Mas, diferente do sensor, sua função é de controlar ou manipular o ambiente físico. Atuadores recebem sinais, pelo dispositivo conectado, e transformam estes sinais em alguma ação física. Por exemplo, um aquecedor de resistência serve como um atuador, pois ele recebe o sinal do dispositivo conectado, e emite calor para aumentar a temperatura do ambiente. Assim como os sensores, o atuador também pode ser configurado, desde que esse processo seja feito a parte, ou seja, o atuador não pode

Figura 2.1 – Modelo de referência para arquitetura IoT




Fonte: Adaptado de (GUTH et al., 2016)

executar um software de configuração nele mesmo. Considerando esse modelo de referência, se o sistema é utilizado somente para medir parâmetros de um ambiente físico, então o sistema não possuirá atuadores.

O dispositivo é um componente de hardware que se conecta aos sensores e atuadores por uma conexão cabeada ou não cabeada. Sua função é processar os dados gerados pelos sensores e controlar os atuadores. No dispositivo, é necessário um software para realizar tais ações com os sensores e atuadores. O dispositivo representa a primeira etapa na qual componentes de softwares são utilizados para manipular os dados produzidos pelos sensores e para controlar os atuadores. Pode-se dizer então que este dispositivo, é a ponte entre o ambiente físico e o mundo digital.

O *gateway* é utilizando em casos nos quais o dispositivo não é capaz de se conectar diretamente ao restante do sistema. Ou seja, caso o dispositivo não possa se comunicar utilizando um protocolo específico ou por algum outro tipo de limitação técnica, o *gateway* é utilizado para compensar essas limitações, oferecendo tecnologias e funcionalidades que facilitam o estabelecimento de uma comunicação entre o dispositivo e o restante do sistema.

Por último, mas não menos importante, o *middleware* de integração IoT é responsável por receber, agregar e entregar para as aplicações finais, os dados os dispositivos conectados.

Também é sua responsabilidade receber comandos da aplicação e repassar esses comandos para os dispositivos, a fim de que eles possam então controlar o atuador para realizar as respectivas ações.

As soluções IoT envolvem diversos dispositivos conectados principalmente por redes sem fio. Considerando a arquitetura em camadas da rede, há diversos protocolos que podem ser utilizados para implementação dos sistemas IoT. Por exemplo, é possível destacar, na camada de aplicação, os protocolos HTTP, MQTT, DDS (GROUP, 2022), AMQP e CoAP, e, na camada física, os protocolos Wifi, Bluetooh, Zigbee (ALLIANCE, 2015), e LoRa (BOR; VIDLER; ROEDIG, 2016).

O foco deste trabalho é estudar protocolos na de camada de aplicação. Considerando o modelo de referência ilustrado na Figura 2.1, o objetivo desses protocolos é prover confiabilidade, desempenho e escalabilidade na comunicação entre os dispositivos e o *middleware* de integração IoT.

### 2.1.2 Segurança dos Dados

Depois de várias décadas de inovação e avanços tecnológicos das redes sem fio e sistemas embarcados, o uso de dispositivos IoT tem sido cada vez mais frequente na sociedade. Pode-se tomar hoje como exemplo os dispositivos de uma casa inteligente, que são capazes de comunicar e interagir entre si de maneira remota, e de forma totalmente autônoma. Também é possível destacar sistemas de rastreamento de veículos, onde dados como localização em tempo real são transmitidos por redes sem fio para os sistemas de monitoramento.

Com isso, surgiu uma grande preocupação em relação à segurança e privacidade dos dados e dispositivos em aplicações de IoT. Apesar de todo avanço observado, a maioria dos dispositivos IoT ainda são considerados vulneráveis a ataques e uma das principais dificuldades de sanar este problema, é manter a simplicidade do dispositivo conforme Dian, Vahidnia e Rahmati (2020) explicam:

> Um dos principais desafios em dispositivos IoT é como implementar políticas de segurança, enquanto se mantém a complexidade do sistema o mais baixo possível. Em geral, dispositivos como relógios são mais fáceis de hackear devido à pobre proteção de criptografia.

Atualmente, a maioria dos dispositivos são passíveis de ataques tanto de software quanto de hardware, porém, ataques ligados ao hardware acontecem de forma bem menos frequente do que os de software. Segundo Xu, Wendt e Potkonjak (2014), é esperado que a maioria dos

ataques de segurança aconteça no software, pois atualmente é mais popular e pode cobrir um grande número de dispositivos e processos.

Como a maioria dos ataques acontecem via software, é de suma importância tomar precauções ao desenvolver uma aplicação IoT. Alguns protocolos e procedimentos podem ajudar a garantir uma confiabilidade maior no dispositivos, tais como: utilização de uma VPN (MASUDUZZAMAN et al., 2020), utilização de protocolos SSH ou TLS(HOZ et al., 2018), dentre outros. Já, do ponto de vista de hardware, é possível contar com protocolos como PUF (WENDT; POTKONJAK, 2014), ARM TrustZone e Security Controller (LESJAK; HEIN; WINTER, 2015) .

## 2.2 O protocolo MQTT

MQTT (*Message Queuing Telemetry Transport*) é um protocolo de transporte de mensagens que utiliza a arquitetura *publish-subscribe*. O protocolo foi criado em 1999 pela organização OASIS, e, desde então, vem recebendo melhorias e atualizações desenvolvidas principalmente pela IBM.

O MQTT vem ganhado uma grande popularidade entre as aplicações IoT que envolvem coletada de dados, superando até mesmo o protocolo HTTP em 2018 de acordo com Koelsch (2022). Grande parte deste crescimento se dá por ser um protocolo que se encaixa muito bem em aplicações IoT, e em ambientes que exigem uma comunicação de máquina para máquina. Esse protocolo também é apropriado para ambientes com conectividade limitada (com taxas de transmissão baixas). O MQTT tem como principal característica ser um protocolo leve, simples e aberto, além de ter um design fácil de ser implementado.

### 2.2.1 A Abordagem *Publish-Subscribe*

Quando se fala sobre comunicação entre dispositivos, é importante entender as diferentes camadas que existem na arquitetura da rede, pois, sem essa organização em camadas e sem conhecer seus respectivos protocolos, a Internet teria muitos problemas de interoperabilidade e desempenho. É justamente a definição clara de comunicação entre diferentes tipos de protocolos que faz com que a Internet funcione hoje da forma que é (ELHADI et al., 2018).

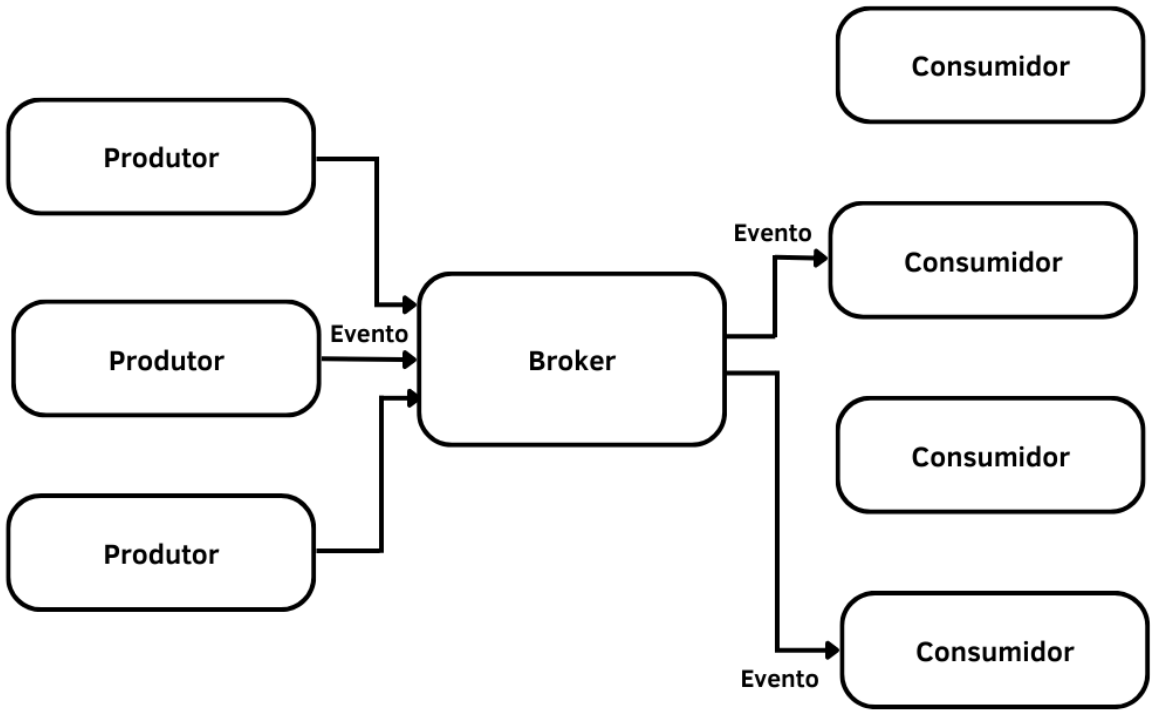
Um dos modelos utilizados para interconectar aplicações distribuídas que vem ganhando bastante destaque é o *publish-subscribe*. Esta abordagem é muito utilizada em aplicações de cunho financeiro, processos automatizados, transporte, dentre outros. O modelo *publish-subscribe* provê uma distribuição de mensagens de um-para-muitos. Isso significa que um dispositivo pode enviar mensagens para vários outros dispositivos distintos. A abordagem *publish-subscribe* utiliza uma comunicação persistente orientada a mensagem, também conhecido como sistemas de enfileiramento de mensagens. Tanenbaum e Steen (2007) define que:

> Sistemas de enfileiramento de mensagens proporcionam suporte extensivo para comunicação assíncrona persistente. A essência desses sistemas é que eles oferecem capacidade de armazenamento de médio prazo para mensagens, sem exigir que o remente ou receptor estejam ativos durante a transmissão da mensagem.

Para isso acontecer, o dispositivo que publicará as mensagens, deve publicá-las em um ou mais tópicos. Estes tópicos são gerenciados por um elemento intermediário, denominado *broker*,

que, por sua vez, tem como função principal, distribuir as mensagens para os devidos clientes que assinam aquele tópico. A Figura 2.2 ilustra o funcionamento do modelo de comunicação *publish-subscribe* no qual todos os produtores realizam o envio de uma mensagem pelo tópico "Evento" e o *broker* transmite essa mensagem para os consumidores que assinam o tópico "'Evento".

Figura 2.2 – Modelo de representação do paradigma *publish-subscribe*



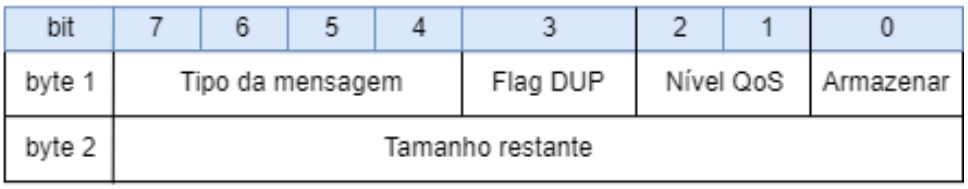

Fonte: (ABDELHALIM, 2022)

O *broker* é a parte principal de todo protocolo *Publish-Subscribe*. O *broker* é um serviço de rede que tem como principal responsabilidade receber toda mensagem que é publicada em um tópico e distribuir essas mensagens corretamente para cada cliente que tenha assinado o respectivo tópico. Além de distribuir as mensagens, o *broker* também tem a função de manter a sessão de cada cliente que esteja conectado, pois cada cliente conectado a um *broker*, precisa persistir sua conexão com o *broker* para continuar publicando e recebendo as mensagens. Como é funcionalidade do *broker* manter a conexão dos clientes, também se torna responsabilidade dele realizar a autenticação e autorização dos clientes. Através do *broker*, é possível criar usuários e configurar o *broker* para que ele funcione somente com usuários autenticados. Mesmo que não seja seguro, o *broker* pode permitir que clientes não autenticados possam se conectar, publicar e receber mensagens.

Outra importante funcionalidade do *broker* é armazenar uma determinada mensagem para entregá-la posteriormente para um cliente que não pode recebê-la em tempo real, por qualquer que seja o problema de conexão, e então, garantir a entrega da mensagem, evitando duplicações dependendo do nível de serviço escolhido pelo cliente (LIGHT, 2021).

A Figura 2.3 ilustra o cabeçalho fixo do protocolo MQTT, constituído por dois bytes. Os quatro primeiros bits representam identificador do tipo de mensagem (se é uma mensagem de conexão, ou *ACK* de conexão, uma publicação ou assinatura, etc). O quinto bit representa a flag para identificar se a mensagem está repetida ou não. O sexto e sétimo bits indicam o nível da qualidade de serviço (QoS). E o último bit é uma flag que indica se o *broker* deve ou não armazenar a mensagem.

Figura 2.3 – Cabeçalho do protocolo MQTT




Fonte: do autor

### 2.2.2 Qualidade de Serviço

O protocolo MQTT utiliza, na camada de transporte, o protocolo TCP para entregas das suas mensagens. Além disso, o desenvolvedor que escolhe o MQTT para implementar sua aplicação pode escolher um nível que serviço que mais se adéqua à aplicação. A qualidade de serviço (QoS – *Quality of Service*) é um acordo feito entre o dispositivo que realizará o envio das mensagens para o *broker*, e o próprio *broker*. Este acordo irá definir o nível de garantia de entrega pra uma mensagem específica. Atualmente, existem 3 classes de QoS no protocolo MQTT: QoS 0, QoS 1 e QoS 2. Cada classe realiza um nível de esforço diferente para entregar as mensagens, ou até mesmo para manter as mensagens no *broker*, caso um assinante não esteja disponível naquele momento para receber a mensagem.

A classe QoS 0 é que faz o menor esforço para entregar a mensagem. Este QoS se assemelha muito ao protocolo de transporte UDP, pois não existe estabelecimento de sessão, nem reconhecimento para garantir a entrega das mensagens, como ilustrado na Figura 2.4. Quem realiza o envio da mensagem também não tem a obrigação de armazenar a mensagem para realizar uma futura retransmissão. As mensagens que são enviadas utilizando este QoS possuem a

maior eficiência de transmissão, mas em compensação, a mensagem pode não ser entregue, por qualquer que seja o problema que possa ocorrer durante a transmissão. Este QoS geralmente é utilizado em aplicações quando não existe uma grande preocupação caso ocorra a perda de alguma mensagem.

Figura 2.4 – Qualidade de Serviço nível 0: Entrega no máximo uma vez


Fonte: The HiveMQ Team (2015)

A classe QoS 1 garante que a mensagem enviada será entregue até o *broker*. O cliente que envia uma mensagem aguarda um *ack* de resposta do *broker*, confirmando que a mensagem foi recebida com sucesso, como mostra a Figura 2.5. Neste método, o cliente mantém a mensagem armazenada, até que o *broker* confirme o recebimento da mesma, para que, em uma eventual falha de conexão, o cliente possa tentar realizar o envio novamente, garantindo assim a entrega da mensagem pelo menos uma vez. É importante notar que neste processo, o cliente pode tentar realizar o envio da mesma mensagem mais de uma vez, o que pode acabar resultando em duplicação da mensagem. Por este motivo, este nível de QoS é conhecido como "*at least once*", pois a mensagem será entregue ao *broker*, pelo menos uma vez, mas pode ser entregue mais de uma vez.

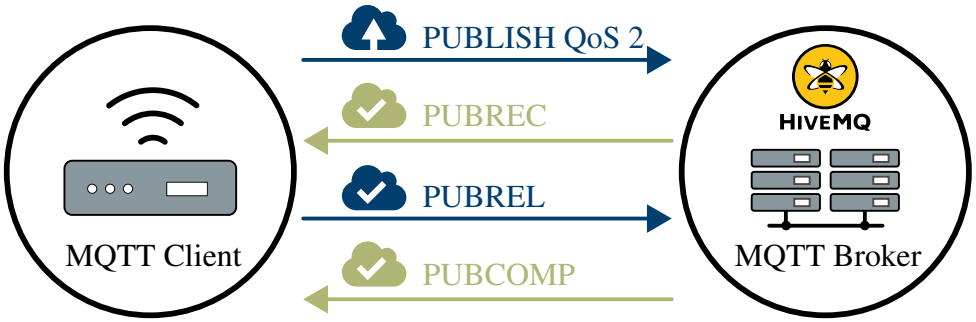
Figura 2.5 – Qualidade de Serviço nível 1: Entrega pelo menos uma vez


Fonte: The HiveMQ Team (2015)

No QoS 2, existe um *hand-shake* de quatro partes, como mostra a Figura 2.6. Este processo não só garante que a mensagem será entregue, mas também garante que não haverá

duplicatas de uma mesma mensagem. Para isso, o cliente envia uma mensagem para o*broker*, ao mesmo tempo que a mantém armazenada até que o processo de publicação chegue ao fim. O *broker*, ao receber a mensagem, a armazena temporariamente também e devolve um *PUBREC* para o cliente. Caso o cliente não receba o *PUBREC* de volta do *broker*, ele reenvia a mensagem, porém passando uma flag "DUP" indicando que esta pode ser uma mensagem duplicada. Se o cliente receber o *PUBREC* com sucesso, então o cliente envia um *PUBREL* para o *broker*, e apaga a mensagem, pois já que o *broker* recebeu a mensagem, não haverá necessidade dele continuar armazenando a mensagem. Quando o *broker* recebe o *PUBREL*, é realizado então o despacho da mensagem para todos os clientes que assinaram aquele tópico. O *broker* então envia uma mensagem *pubcomp* para cliente e apaga a mensagem que foi armazenada temporariamente por ele. Caso o *broker* não tenha recebido o *PUBREL*, o cliente reenvia o *PUBREC*. Por fim, quando o cliente recebe o *pubcomp* o processo de publicação foi completado, e só então, ele apaga a mensagem por completo.

Figura 2.6 – Qualidade de Serviço nível 2: Entrega exatamente uma vez




Fonte: The HiveMQ Team (2015)

### 2.2.3 Segurança

O protocolo MQTT possui os atributos de *username* e *password* para serem utilizados quando for iniciar uma conexão. O cliente tem a opção de utilizar esse mecanismo de autenticação quando for se conectar com o *broker* do MQTT.

O atributo *username* é uma string UTF-8. A senha é atributo binário com no máximo 65535 bytes. Conforme explicado em (HIVEMQ, 2015), a especificação do protocolo MQTT permite que seja possível o cliente enviar o *username* sem uma senha, porém não é possível enviar uma senha sem um *username*. É importante ressaltar que, ao utilizar o mecanismo de login do MQTT, a informação de login não é enviada de forma criptografada para o *broker*. Isso pode resultar em uma falha de segurança abrindo brechas para possíveis ataques.

O protocolo MQTT utiliza o protocolo de transporte TCP. Por padrão, as conexões TCP não utilizam nenhuma comunicação encriptada. Para realizar a encriptação das mensagens MQTT, os *brokers* podem utilizar o protocolo TLS (DIERKS; ALLEN, 1999) sobre o protocolo TCP, ou seja, a mensagem é criptografada e depois encapsulada em um segmento TCP. Utilizando o protocolo TLS, o mecanismo de autenticação se torna protegido, pois toda informação é criptografada desde o processo de estabelecer uma conexão com o *broker*, até mesmo no envio e recebimento de mensagens.

É preciso ficar atento ao utilizar TLS no MQTT, pois incluir o protocolo adiciona um custo adicional no uso de CPU, e também aumenta o *overhead* de comunicação. Enquanto o aumento do uso de CPU é geralmente insignificante pelo *broker*, isso pode acabar sendo um problema para dispositivos IoT que não possuem uma capacidade de processamento maior. Dispositivos que não foram criados para atividades de computação intensa podem acabar ocasionando falhas.

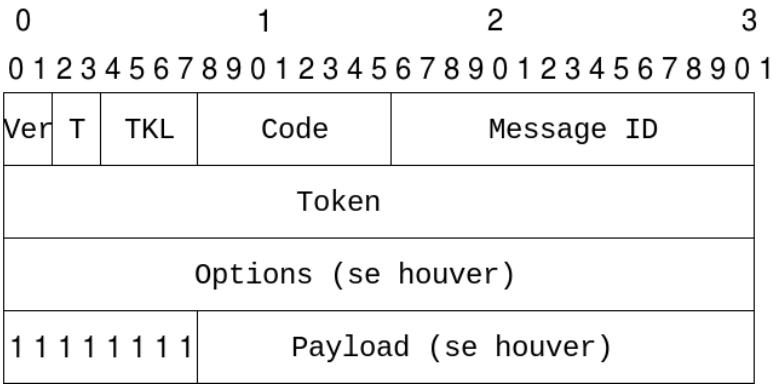
## 2.3 O protocolo CoAP

O CoAP (*Constrained Application Protocol*) é um protocolo conhecido como "*report-only*", ou seja, ele é recomendado para aplicações cujos dispositivos somente reportam dados para o servidor. CoAP é um protocolo capaz de atuar em ambientes com limitações tais como: baixa largura de banda, dispositivos com energia limitada, ou em conexões com grande probabilidade de eventuais perdas de pacotes, e até mesmo dispositivos com limitação hardware. Esse protocolo é ideal para sistemas de automação residencial onde a conexão entre os componentes de uma mesma rede precisam ser leves e ter um baixo custo.

Um destaque do CoAP, quando se fala de protocolos de aplicação IoT, é que ele tem uma interface do tipo *RESTFUL* (FIELDING, 2000) ou seja, é possível realizar requisições HTTP, do tipo *GET*, *POST*, *PUT* E *DELETE*. O próprio protocolo já cuida disso e não é preciso que o desenvolvedor replique esse comportamento manualmente conforme mencionado em (UGRENOVIC; GARDASEVIC, 2015). Ao utilizar o paradigma *REST*, o protocolo CoAP realiza uma comunicação transiente orientada a mensagem. Mensagens transientes são mensagens enviadas somente em tempo real e não são armazenadas em nenhum local. O receptor da mensagem somente receberá a mensagem se ele estiver online (TANENBAUM; STEEN, 2007).

A diferença chave entre o protocolo HTTP e o CoAP está na camada de transporte. O Protocolo HTTP utiliza uma conexão orientada pelo protocolo TCP, enquanto o CoAP foi construído para funcionar utilizando protocolo UDP. O Protocolo CoAP realiza *handshaking*, e por utilizar o protocolo UDP na camada de rede, esse handshaking é mais leve e fácil de ser implementado em microcontroladores. O cabeçalho do protocolo CoAP utiliza apenas 4 *bytes* (LUDOVICI; MORENO; CALVERAS, 2013), como representado na Figura 2.7. São 2 *bits* alocados para informar a versão do CoAP, 2 *bits* para indicar o tipo da mensagem, 4 *bits* reservados para indicar o comprimento de token (TKL), 8 *bits* para indicar a classe da mensagem (que pode ser request, response de sucesso ou erro) e 16 *bits* utilizados para identificar mensagens duplicadas através do QoS Confirmável ou não confirmável.

Figura 2.7 – Cabeçalho do CoAP




Fonte: (SILVA FELIPE FADUL DE CARVALHO, 2019)

Outro fator que diferencia o CoAP do HTTP é que o CoAP tem suporte a *multicast*, ou seja, ele consegue enviar uma resposta para mais de um cliente de uma vez, diferentemente do HTTP, que não possui esta funcionalidade.

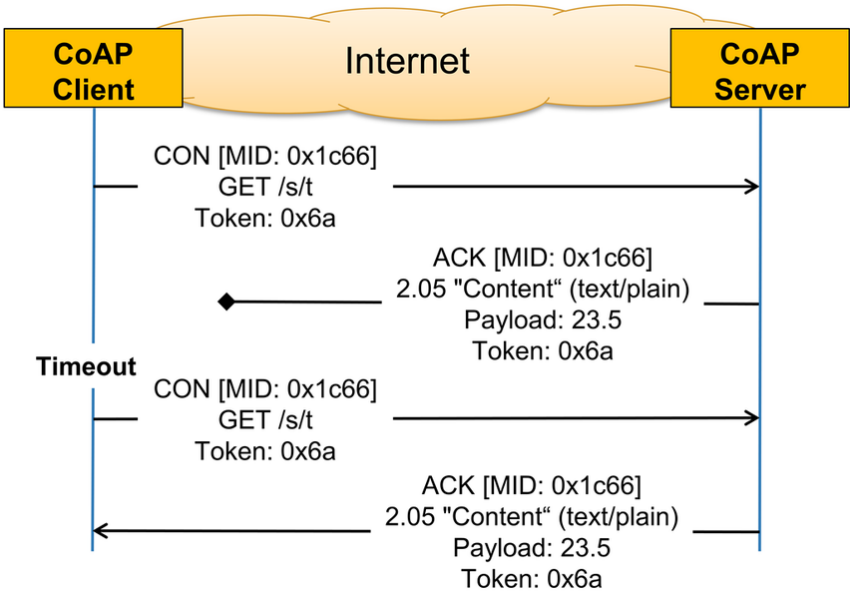
A ideia principal do CoAP em seguir o estilo arquitetural RESTFull é utilizar os recursos e funcionalidades já existentes para WEB. Desta forma, o CoAP não requer novas tecnologias, APIs, ou serviços, simplesmente reaproveita aqueles que já são largamente utilizados. Além disso, esse o CoAP se torna viável para o uso em ambientes mais limitados, com o intuito de economizar recursos e fazer com que o servidor seja leve o bastante para ser executado em sistemas que possuem uma limitação de hardware de baixo nível de processamento (UGRENOVIC; GARDASEVIC, 2015).

### 2.3.1 Entrega Confiável

Enquanto outros protocolos como o MQTT possuem classes de serviço (níveis de QoS) que permitem a verificação do recebimento de uma mensagem, o CoAP, utilizando UDP, possui apenas uma maneira de checar o recebimento de uma mensagem. Para realizar essa checagem, o protocolo CoAP utiliza o atributo CON (*confirmable*) nas suas mensagens. O atributo CON nada mais é que uma mensagem do tipo *acknowledgement* confirmando que a mensagem foi entregue. Por padrão, ao enviar uma mensagem (ou *request*) para o servidor, o CoAP inicia uma contagem de *timeout* aguardando o *acknowledgement*. Se o servidor não responder antes do esgotamento do temporizador com o mesmo *Message ID*, o cliente reenvia o *request* novamente, até que o *acknowledgement* de resposta do servidor seja entregue com sucesso para o cliente (SILVA FELIPE FADUL DE CARVALHO, 2019).

A Figura 2.8 ilustra como é realizada a entrega de uma mensagem de forma confiável. O cliente CoAP, ao realizar uma requisição para o servidor CoAP, passa no seu cabeçalho um atributo *CON* com a chave da requisição. O servidor CoAP, ao receber a requisição, retorna um *ACK* para o cliente. Caso o cliente não receba esse *ACK* do servidor, ele realiza novamente o request, informando a mesma chave no *CON* que havia sido enviada anteriormente.

Figura 2.8 – Exemplo de confirmação de recebimento de mensagem no protocolo CoAP.




Fonte: (ISHAQ et al., 2016)

### 2.3.2 Segurança no CoAP

O protocolo CoAP utiliza o *Datagram Transport Layer Security* (DTLS) como principal ferramenta de segurança para comunicação entre seus nós. DTLS é um protocolo que realiza uma troca de mensagens para estabelecer uma conexão segura, para isso, DTLS é baseado em PSK(*Pre-Shared Key*), RPK (*Raw Public Key*) e certificado de segurança. PSK é o conjunto de chaves que foram estabelecidas que permite identificar quais outros agentes podem estabelecer conexões. RPK utiliza de um par assimétrico de chaves sem certificado. Desta forma, o dispositivo possui uma identificação calculada previamente pela chave, e uma lista de nós identificados indicando quais nós ele pode se comunicar. Para todo o processo de segurança funcionar corretamente, todos os nós devem ser clientes DTLS, desta forma, é possível realizar o *hand-shake* para estabelecer a conexão e, só depois, começar a transmitir as mensagens.

Além disso, é de suma importância que durante a interação entre o cliente e o servidor (quando utilizando DTLS), a requisição realizada deve possuir sessão, um *timestamp* identifi-

cando a data/hora do *request* e um *Message ID* correspondente. Somente com esses atributos, será possível garantir a eficácia na segurança do protocolo. O mesmo deve ser feito para a resposta do *request* (SILVA FELIPE FADUL DE CARVALHO, 2019).

## 2.4 O protocolo AMQP

Criado em 2003 por John O'Hara, o protocolo AMQP (*Advanced Message Queuing Protocol*) foi desenvolvido originalmente para ser utilizado no mercado financeiro. Seu principal propósito era permitir a interoperabilidade entre dispositivos e plataformas. O projeto do AMQP foi estabelecido como um projeto aberto e colaborativo. Muitas empresas participaram de seu processo de desenvolvimento, tais como: Cisco, Microsoft, Redhat e Rabbit Technologies, empresa na qual se iniciou a criação do RabbitMQ[1].

O protocolo é orientado a mensagens assim como MQTT. Trabalha com controle de fluxo e balanceamento de carga. Seu diferencial é o tratamento das mensagens, armazenando-as em uma fila (*queue*) de acordo com suas chaves de tópicos, e a entrega dessas mensagens quando os consumidores se encontram disponíveis. As mensagens permanecem armazenadas na fila, até que o consumidor consiga se conectar ao serviço e consuma as mensagens.

O AMQP possui duas configurações em relação à garantia de entrega de suas mensagens, sendo elas: *at-most-once*, ou seja, a mensagem poderá ser entregue no máximo uma vez e mesmo assim, não há garantia de entrega; e *at-least-once*, que garantirá que a mensagem será entregue pelo menos uma vez, podendo haver duplicatas da mensagem (RABBITMQ, 2007).

### 2.4.1 A abordagem Publish-Subscribe do AMQP

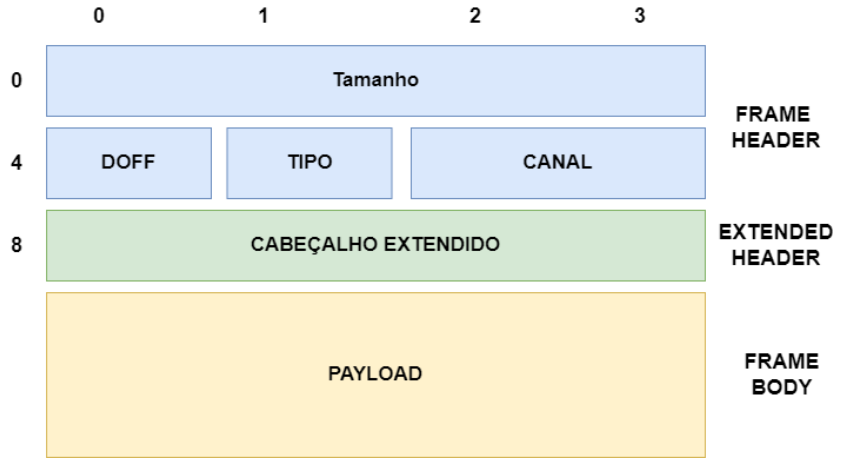
Apesar de semelhante, o AMQP tem algumas particularidades que se diferenciam do protocolo MQTT. A principal está no sistema de filas. No protocolo AMQP, o *broker* deve implementar um componente denominado "*exchange*". A principal função da *exchange* é colocar as mensagens nas suas respectivas filas. Com essa mudança, o protocolo AMQP passa a ser mais pesado que o concorrente MQTT.

A Figura 2.9 ilustra o cabeçalho do protocolo AMQP. Nela é possível observar que o protocolo possui um cabeçalho de 8 bytes. Os bytes 0 a 3 do cabeçalho representam o tamanho completo do *frame*, na forma de um número inteiro de 32 bits. O quarto byte, conhecido como DOFF, representa onde os dados do *frame* começam. O quinto byte indica o tipo do *frame*, com isso ele consegue identificar o formato e propósito do *frame*. Por exemplo, o código 0x01 indica que o *frame* é um SASL. Os últimos 2 bytes do cabeçalho informam o número do canal do *frame*. O *extended header* é utilizando quando o *frame* utiliza SASL, caso contrário, é

[1] RabbitMQ é um *broker open-source* bastante utilizado desenvolvedores IoT utilizando AMQP. Ele está disponível em <https://www.rabbitmq.com/>.

ignorado. E, por último, o *frame body* é o *payload* em si, contendo a informação da mensagem (STANDARD, 2012).

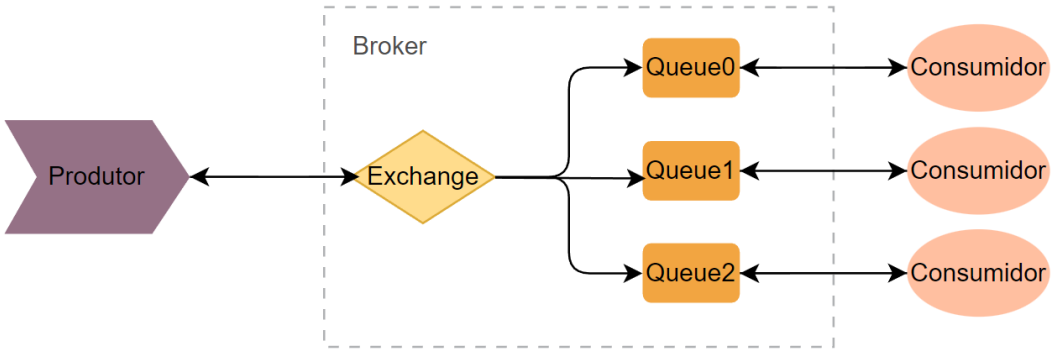
Figura 2.9 – cabeçalho do protocolo AMQP.




Fonte: do autor

A Figura 2.10 ilustra o funcionando do protocolo AMQP, na qual o produtor se conecta a um *broker*, enquanto cada um dos consumidores se conecta a uma fila distinta da *exchange*. O *broker* do protocolo AMQP tem como responsabilidade receber as mensagens enviadas pelo produtor e transmiti-las para os consumidores. Porém, esse processo de transmitir a mensagem, não é realizado diretamente pelo *broker*. Esse processo é realizado pela *exchange*. A *exchange* irá pegar as mensagens recebidas pelo broker e encaminhá-las para as filas (*queues*). As filas são entidades presentes dentro do *broker* que armazenam as mensagens, para depois realizar o envio das mesmas para os consumidores que assinam esta determinada fila. Um *broker* pode conter várias filas. Vários consumidores podem estabelecer várias conexões simultâneas com vários *brokers* e assim, assinar várias filas distintas.

Figura 2.10 – Desenho arquitetural do protocolo AMQP.

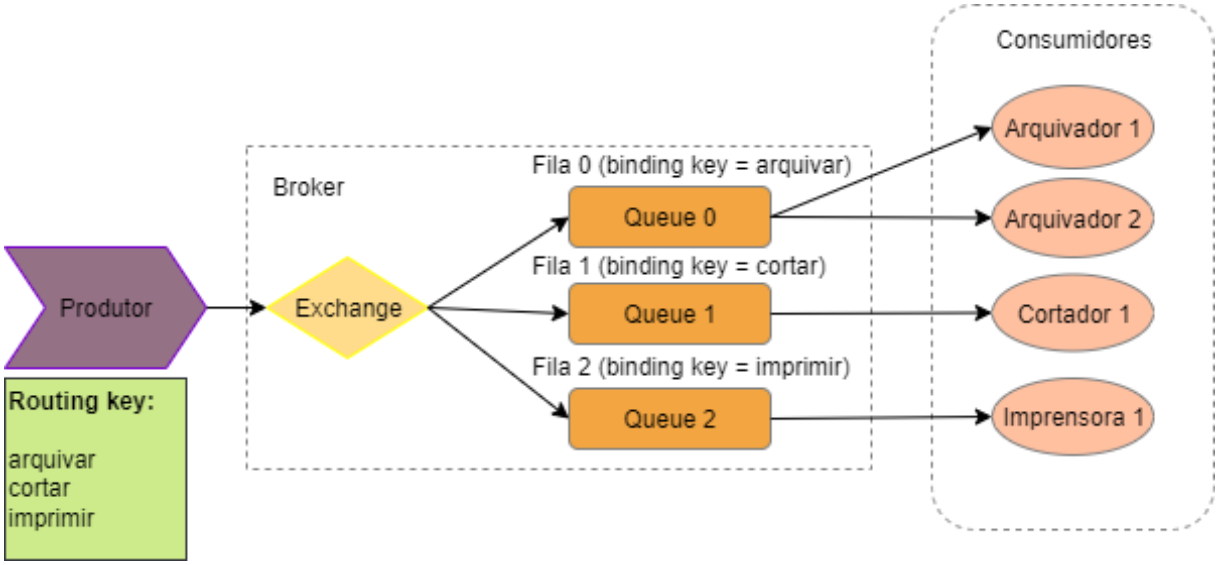



Fonte: do autor

Ao publicar uma mensagem, o produtor (*sender*) precisa especificar qual broker irá realizar o envio da mensagem e também para qual fila a mensagem deverá ir. Quando a mensagem é enviada para o *broker*, a *exchange* encaminhará a mensagem para a(s) fila(s) especificadas no cabeçalho da mensagem. Uma vez que a mensagem estiver na fila, ela será entregue sempre que os consumidores que a assinam estiverem online.

O protocolo AMQP especifica 3 métodos de operação da *exchange*. O método mais comum é o *direct*. Nele, a *exchange* realiza a entrega das mensagens às filas de acordo com a *rounting key* da mensagen. Ou seja, o produtor especifica quais filas irão receber as mensagens, e desta forma, a *exchange* realiza a entrega das mensagens para as filas que o produtor especificou no cabeçalho da mensagem. Esta é a *exchange* mais utilizada pelo protocolo, pois com ela é possível dividir responsabilidades e ações a serem executadas pelos consumidores.

Figura 2.11 – Representação arquitetural da *exchange direct*




Fonte: do autor

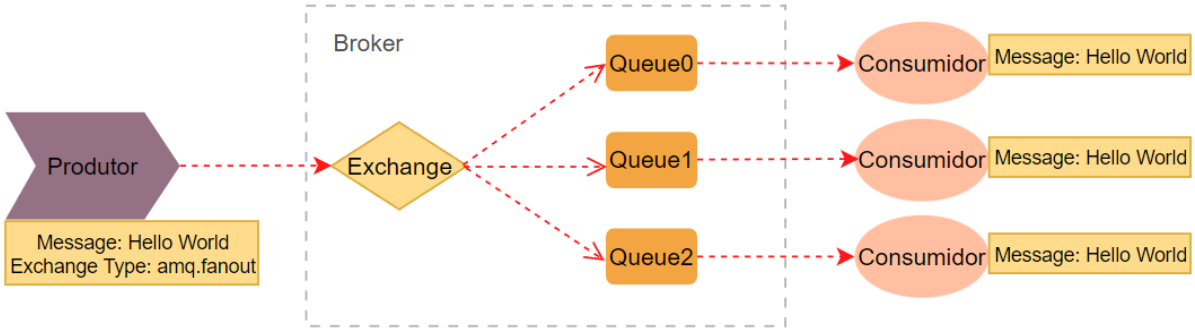
A Figura 2.11 ilustra o funcionamento do protocolo quando o produtor realiza o envio de três mensagens, cada um para uma fila distinta da *exchange*, simulando um sistema de gerenciamento de arquivos. O produtor, que possui os arquivos (que também pode ser denominado como mensagem), deseja enviar os mesmos para o devido processamento. Então cada arquivo é enviado para para uma fila, onde respectivamente os consumidores processarão os arquivos, agindo como atuadores.

Outro método de operação da *exchange* é o *fanout*. Neste caso, o produtor, ao realizar o envio da mensagem, especifica em seu cabeçalho que o tipo da *exchange* será *fanout*. Quando a *exchange* receber a mensagem, irá replicá-la para todas as fila existentes no *broker*, de forma

que todos os consumidores de qualquer fila deste *broker* receberão a mensagem. Um bom exemplo de uso para este tipo de *exchange* é quando é necessário realizar um *broadcast* de uma mudança de configuração para todas as filas, fazendo com que seus respectivos consumidores repliquem as alterações.

A Figura 2.12 ilustra o comportamento do *fanout*, na qual o produtor ao realizar o envio de uma mensagem para o *broker*, especifica o atributo *exchange type*, informando que será um *fanout*. Quando o *broker* recebe a mensagem e a *exchange* vê que o tipo é *fanout*, ele automaticamente realiza o *broadcast* da mensagem para todas as filas.

Figura 2.12 – Desenho arquitetural do protocolo AMQP utilizando *fanout*

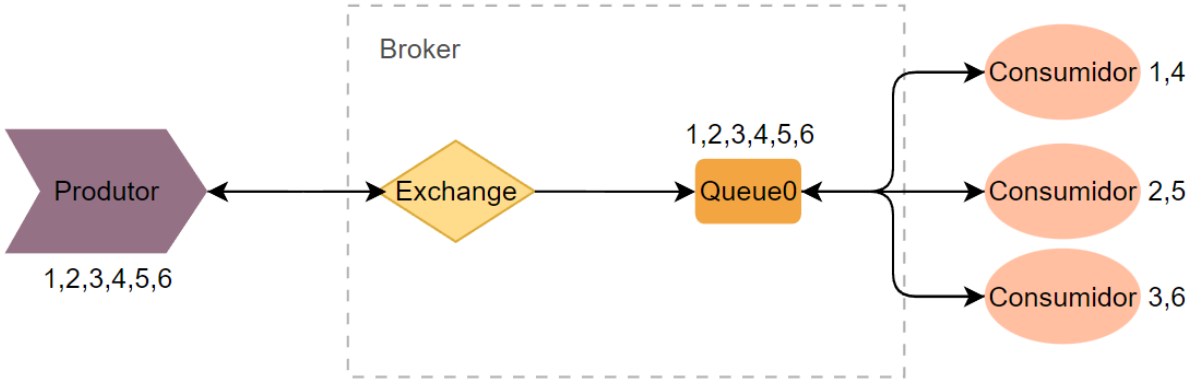



Fonte: do autor

O terceiro modo de operação da exchange é o modo concorrência. Quando se tem múltiplos consumidores associados a uma mesma fila, o AMQP irá rotacionar qual consumidor receberá a mensagem. Vale ressaltar que, durante esse processo, não haverá duplicidade nas mensagens, ou seja, nenhum consumidor receberá a mesma mensagem que outro consumidor recebeu.

A Figura 2.13 ilustra o comportamento do AMQP, quando o produtor realiza o envio de seis *payloads* para uma única fila da *exchange*. A fila recebe todas as mensagens e, ao distribuir as mensagens, rotaciona cada mensagem em sequência para cada consumidor. O primeiro consumidor receberá o *payload* "1", o segundo consumidor o *payload* "2", o terceiro consumidor o *payload* "3", então o primeiro consumidor receberá o próximo *payload*, dando sequência assim na rotação de *delivery*, conforme explicado em RabbitMQ (2007).

Figura 2.13 – Desenho arquitetural do protocolo AMQP com concorrência




Fonte: do autor

### 2.4.2 Qualidade de Serviço

Assim como no protocolo MQTT, o AMQP também utiliza protocolo TCP, que garante entrega de pacotes e retransmissão dos mesmos em caso de falha a nível de camada de rede. Para garantir as entregas a nível de aplicação, o protocolo AMQP também utiliza uma mecanismo de provisão de QoS (*Quality of service*). Porém, neste protocolo, estão definidas somente duas classes de QoS: *Unsettle* e *Settle*.

No QoS 0 (*Unsettle*), o produtor dispara a mensagem para um broker e imediatamente apaga a mensagem. Neste processo, caso a mensagem não chegue até o broker, ela será perdida e não poderá ser retransmitida.

No QoS 1 (*Settle*), o produtor dispara a mensagem, mas fica aguardando uma resposta de confirmação do *broker*. Somente depois que o *broker* confirma o recebimento da mensagem, enviando um *acknowledgment* de volta para o produtor, é que o produtor apaga a mensagem. Se o produtor não receber um *acknowledgment* em um determinando intervalo de tempo após o envio da mensagem, ele re-transmite a mensagem novamente para o *broker*, podendo ocasionar em duplicações de mensagens, ao mesmo tempo garantindo a entrega da mensagem pelo menos uma vez.

### 2.4.3 Segurança

Camadas de segurança são usadas para estabelecer uma autenticação ou transporte encriptado sobre o protocolo AMQP. É possível aplicar várias camadas de proteção, de forma que elas funcionem em paralelo. Por exemplo, uma camada de segurança utilizada para realizar

autenticação pode ser conectada a outra camada de segurança estabelecida para realizar alguma encriptação.

No protocolo AMQP, *containers* é o nome dado para uma aplicação que pode enviar e receber mensagens. Com o uso de *containers*, o protocolo AMQP permite uso de multiplexação, ou seja, um cliente pode receber uma mensagem de uma fila e enviar mensagens para outra fila na mesma conexão de rede simultaneamente. Para instanciar um *container*, o protocolo utiliza uma conexão TCP. Para essa conexão TCP, é realizado um *handhsake* entre os *containers* declarando o uso de TLS/SSL e um *handshake* de autorização/autenticação baseado em SASL (*Simple Authentication and Security Layer*).

Em geral, o SASL simples normalmente é utilizado para transmitir as credenciais de login e senha para o gerenciador de filas para fins de autenticação. Geralmente, o servidor utilizando SASL manda uma lista de mecanismos de autenticação aceitáveis para o cliente. O cliente utilizando SASL pode decidir qual mecanismo utilizar de forma a melhor satisfazer os requisitos. Uma vez que o cliente escolhe o mecanismo de autenticação e realiza o *handshake* com o servidor, é estabelecida uma conexão com o servidor através do *container*.

# 3 ANÁLISE COMPARATIVA

Fazer a escolha pelo protocolo IoT mais adequado envolve avaliar as funcionalidades que cada protocolo possui e considerar vários fatores relacionados a confiabilidade, desempenho, segurança, entre outros relevantes. É preciso ponderar quais funcionalidades a aplicação necessita e quais aspectos o arquiteto da aplicação estará disposto comprometer em função das funcionalidades e mecanismos escolhidos.

Antes de decidir qual protocolo utilizar, é preciso avaliar fatores como: a) em que tipo de *hardware* a aplicação será executada; b) qual a qualidade de rede disponível; c) se a aplicação poderá receber mensagens repetidas; d) e, principalmente, como deve ser a estrutura de troca de mensagens dentro do sistema IoT.

Neste trabalho, foi realizada uma revisão bibliográfica a fim de se elaborar uma análise comparativa sobre protocolos IoT de camada de aplicação. O resultado desta análise pode auxiliar um arquiteto de soluções a escolher um protocolo de forma mais adequada ao cenário de aplicação.

## 3.1 Metodologia

Para realizar os estudos e levantamento de dados que embasam este trabalho, que consiste em apresentar uma análise qualitativa e desempenho dos protocolos IoT. Do ponto de vista de objetivo, este trabalho se caracteriza como pesquisa exploratória. Em se tratando dos procedimentos técnicos este trabalho pode ser delineado como um pesquisa bibliográfica Gil et al. (2002). Este tipo de pesquisa é desenvolvida a partir de leituras de artigos científicos, livros entre outras fontes que foram reconhecidas previamente pelo meio acadêmico, e que podem contribuir no que se refere aos aportes teóricos do que está sendo discutido. Consiste ainda em utilizar as propostas apresentadas nestes trabalhos como ferramentas para análise e até mesmo comparação entre as resultados (GIL et al., 2002).

Para este estudo, foram escolhidos os três protocolos IoT na camada de aplicação mais populares: MQTT, CoAP e AMQP (AL-MASRI et al., 2020). Além desses, há mais três outros protocolos de aplicação entre os mais populares:

- O próprio HTTP, que foi excluído deste estudo por não ser específico para aplicações IoT.

- O protocolo *Data Distribution Service* (DDS), que é focado no compartilhamento descentralizado de dados e não é tão utilizado como os demais protocolos mencionados, tanto que não é suportado pelos grandes provedores públicos de plataformas IoT em Nuvem.

- O protocolo *Extensible Messaging and Presence Protocol* (XMPP), originalmente conhecido como Jabber, não foi projeto para aplicações IoT, e necessita do uso de extensões para esta finalidade.

Durante a realização deste trabalho, houve dificuldade para encontrar documentos e estudos comparativos de performance dos protocolos escolhidos. Cada autor realizou seus experimentos em ambientes distintos com métricas e variáveis diferentes. Encontrar estudos que corroborassem os resultados mostrou ser uma dificuldade durante este estudo.

A análise qualitativa tem como principal objetivo demonstrar as principais diferenças dos protocolos descritos previamente, comparando não só as funcionalidades existentes de cada um, mas também as especificações técnicas. Desta forma, objetiva-se auxiliar arquitetos de software na escolha de um protocolo para seu projeto IoT.

Para a análise qualitativa, foram levadas em consideração as funcionalidades que os protocolos possuem. Foram consultados documentos técnicos e científicos que descrevem as funcionalidades e especificações dos protocolos, incluindo aspectos tais como confiabilidade e segurança, e também trabalhos que descrevem cenários de uso indicados para cada protocolo. Após comparar as funcionalidades de cada protocolo, foi construída uma tabela comparativa das funcionalidades e especificações dos protocolos.

A análise de desempenho tem como objetivo expor dados referentes ao desempenho dos protocolos. Para essa comparação de desempenho, foram levadas em consideração as seguintes métricas: latência, *overhead*, transferência e consumo energético. Nesta etapa, foram escolhidos artigos que realizaram estudos sobre o desempenho com os protocolos mencionados anteriormente. Desta forma, a seção tem como intuito prover uma comparação da performance dos protocolos por meio de revisão bibliográfica sobre o assunto.

## 3.2 Análise Qualitativa

Realizando um comparativo entre os três protocolos citados anteriormente, é possível notar que o protocolo CoAP é o mais peculiar, na medida em que apresenta aspectos arquiteturais completamente diferente dos demais. Sua Arquitetura REST pode trazer um desen-

volvimento significativamente mais rápido, já que a arquitetura REST é bastante utilizada no mercado atualmente. Levando em consideração a curva de aprendizado que um time de desenvolvimento levaria para desenvolver uma aplicação IoT, utilizar um protocolo que oferece formas de interação por meio de requisições HTTP, além de ser confortável para a maioria dos desenvolvedores, também tira a responsabilidade dos mesmos de ter que tratar manualmente estes comportamentos. Note que, apesar dos protocolos MQTT e AMQP não possuírem funcionalidades de uma arquitetura REST, nada impede o arquiteto da aplicação de estruturar as mensagens que serão trocadas nestes protocolos, para reproduzirem esta funcionalidade.

Os protocolos MQTT e AMQP auxiliam a comunicação entre servidor o servidor e o cliente somente através de um *broker*. Já com o protocolo CoAP, cada nó interage diretamente com o servidor. Apesar de cortar intermediários, protocolo CoAP leva uma desvantagem quando precisa se comunicar com vários servidores de uma vez, já que será necessário uma requisição individual para cada servidor. Já nos protocolos MQTT e AMQP, caso um cliente deseje fazer um *broadcast* para todos os outros clientes, basta que todos estes clientes estejam assinando um tópico específico (MQTT) ou realizar um *fanout* (AMQP). O Broker se encarregará de distribuir a mensagem para todos os clientes conectados.

No requisito qualidade de serviço, o protocolo MQTT se destaca dos demais protocolos por possuir três níveis de QoS. Além de garantir a entrega da mensagem, o MQTT também pode impossibilitar duplicações de mensagens. Os protocolos CoAP e AMQP, quando realizam troca de mensagens utilizando uma qualidade de serviço que garante a entrega das mensagens, podem acabar gerando duplicatas da mensagens.

A Tabela 3.1 apresenta um resultado geral deste estudo, com uma análise de vários critérios que caracterizam cada um dos protocolos. Essa análise pode ajudar arquitetos e desenvolvedores de aplicações IoT na tomada de decisão referente a escolha de um protocolo. No que diz respeito a persistência de conexão, o protocolo CoAP não possui esta funcionalidade, de maneira que as requisições podem ser realizadas a qualquer momento sem antes estabelecer uma sessão entre cliente e servidor. Já os protocolos MQTT e AMQP necessitam criar uma sessão de conexão entre cliente e servidor para que possa acontecer a troca das mensagens. O protocolo MQTT permite apenas uma conexão por vez entre um cliente e servidor, ou seja, caso o cliente queira se conectar a outro *broker*, a conexão atual precisa ser fechada, para que uma nova possa ser estabelecida com outro *broker*. Já o protocolo AMQP permite multiplexação, ou seja, o cliente pode se conectar com mais de um *broker* ao mesmo tempo.

Tabela 3.1 – Comparativo entre os protocolos

| **Critério** | **MQTT** | **CoAP** | **AMQP** |
|---|---|---|---|
| **Nome Completo** | Message Queuing Telemetry Transport | Constrained Application Protocol | Advanced Message Queuing Protocol |
| **Ano de lançamento** | 1999 | 2010 | 2003 |
| **Arquitetura** | Utiliza a arquitetura Publish Subscribe | Utiliza a arquitetura REST ou Publish Subscribe | Utiliza a arquitetura Publish Subscribe |
| **Qualidade de Serviço** | QoS 0 - No máximo uma (Fire-and-Forget), QoS 1 - Pelo menos uma, QoS 2 - Exatamente uma. | Mensagens confirmadas, ou não confirmadas. | *Unsettle* - No máximo uma, e *Settle* - Pelo menos uma. |
| **Modo de Mensagens** | Uso apenas assíncrono | Uso Assíncrono e Síncrono. | Uso apenas assíncrono |
| **Protocolo de camada de transporte** | TCP | UDP | Geralmente TCP, mas também suporta outros protocolos (SCTP, Pipes, etc) |
| **Tamanho do cabeçalho** | De 2 a 5 bytes | Cabeçalho fixo de 4 bytes | Cabeçalho fixo de 8 bytes |
| **Suporte de persistência de conexão** | Oferece suporte uma sessão de conexão por vez | Não possui suporte de persistência de conexão | Oferece suporte a mais de um *broker* ao mesmo tempo |
| **Rotulagem de Mensagem** | Não tem esse recurso. | Ele fornece adicionando rótulos às mensagens. | Sempre adiciona rotulagem a mensagem informando a *exchange* |
| **Protocolos de Segurança** | TlS/SSL | DTLS, IPSec | TLS/SSL, SASL |

Os protocolos MQTT e AMQP são assíncronos, ou seja, produtores e consumidores são desacoplados um do outro. Isso implica que, para receber uma mensagem, o consumidor não precisa estar em execução (disponível) quando a mensagem é enviada pelo produtor. O envio e recebimento das mensagens são eventos independentes. Já o protocolo CoAP pode funcionar de modo síncrono e assíncrono. Por padrão, requisições REST são síncronas, ou seja, ao realizar uma requisição, o cliente aguarda uma resposta do servidor. Para funcionar de forma assíncrona, quando um cliente do protocolo CoAP realiza uma requisição, o cliente ao receber um *ACK* do servidor, pode liberar os recursos necessários para realizar uma nova requisição, enquanto aguarda a resposta da primeira. É importante ressaltar que esta funcionalidade somente estará

disponível quando as requisições realizadas utilizarem a qualidade de serviço de mensagens confirmadas, para que o servidor possa confirmar o recebimento da requisição para o cliente.

## 3.3 Análise de Desempenho

Além de analisar as características funcionais de cada protocolo, o arquiteto envolvido no projeto de uma aplicação IoT também precisa se preocupar com o gerenciamento de recursos. Aspectos como *overhead*, largura de banda e latência podem se tornar críticos dependendo do cenário de aplicação de IoT.

### 3.3.1 Overhead

Naik (2017) realizou um estudo de desempenho apontando as principais diferenças entre alguns protocolos de IoT. Um dos critérios analisados foi a relação do tamanho da mensagem em relação ao *overhead* da mensagem. Em seu estudo, Naik (2017) demostra que o protocolo CoAP possui o menor *overhead* dentre os três protocolos. Já o protocolo AMQP possui o maior *overhead*. Ele também ressalta que:

> Os métodos de avaliações foram baseados em componentes estáticos e sobre evidências empíricas e revisões bibliográficas. Apesar disso, não foram consideradas as condições de rede e *overheads* causados pela retransmissão de pacotes, o que poderia mudar o resultado das comparações.

Thangavel et al. (2014) realizaram um experimento comparando MQTT e CoAP. Neste estudo, os resultados se mantiveram coerentes com o mencionado anteriormente, demonstrando que, quando a taxa de perda de pacotes é baixa, o CoAP gerou menos *overhead* que MQTT para todos os tamanhos de mensagem.

A principal relação disso se dá pelo protocolo na camada de transporte. Protocolos MQTT e AMQP utilizam TCP, logo, possuem um *overhead* maior em relação ao CoAP. O protocolo MQTT consegue ter um *overhead* menor que o AMQP pelo fato de seu cabeçalho possuir apenas 2 bytes de tamanho. Como o CoAP utiliza protocolo de transporte UDP, mesmo que o tamanho de seu cabeçalho seja maior que o do MQTT, seu *overhead* acaba se tornando menor.

No estudo realizado por Thangavel et al. (2014), ao utilizar o protocolo CoAP com confirmação de entrega, os resultados podem ser diferentes. Como o CoAP utiliza UDP, a chance de ocorrer perda da mensagem, se torna maior a medida que o tamanho da mensagem aumenta. Já o MQTT e AMQP por utilizarem TCP possuem uma chance menor de sofrerem com perda

da mensagem durante a transmissão. Neste cenário, os testes apontam que a retransmissão da mensagem, influencia diretamente no *overhead*. E quanto maior a mensagem, maior o overhead no CoAP quando utilizado com confirmação de entrega.

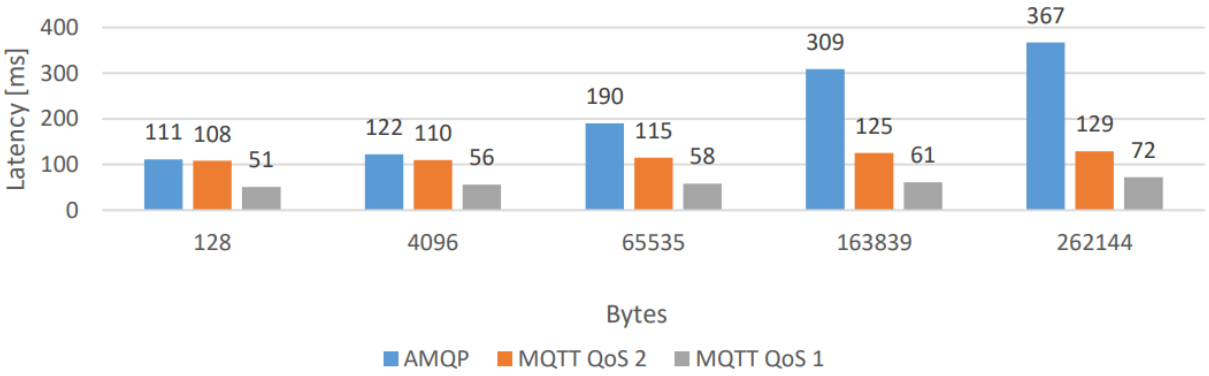
### 3.3.2 Latência

Quando se avalia a latência na entrega das mensagens, a escolha do protocolo de transporte UDP se mostra mais uma vez ser um benefício relevante. O protocolo CoAP possui a menor latência e menor uso de banda de rede. Transmissões UDP reduzem significativamente o tempo de respostas na rede. Conforme mencionado por Naik (2017), vários estudos experimentais demonstram que mesmo utilizando o protocolo MQTT no QoS 1 ou 2, o consumo de banda ainda é maior do que no protocolo CoAP transmitindo o mesmo *payload* em mesma condição de rede. Mesmo comparando MQTT QoS 2 com CoAP utilizando confirmação de mensagem, o Protocolo MQTT chega a utilizar quase o dobro de banda de rede que o CoAP utiliza. Isso acontece pelo fato do QoS 2 utilizar um *handshake* de 4 etapas. O protocolo AMQP tem a maior latência e utilização de banda de rede devido aos vários serviços que o protocolo oferece.

No trabalho apresentado em (LARMO et al., 2018), os autores compararam a latência do protocolo CoAP com o MQTT. Neste caso, o protocolo CoAP foi utilizado sem a confirmação de entrega. Quanto ao MQTT, foi utilizado uma média de latência entre as três qualidades de serviço. Os resultados demonstram que o protocolo CoAP possui menor latência que o protocolo MQTT. Também é ressaltado que a alta latência do protocolo MQTT está relacionada ao alto consumo de largura de banda.

Lindén (2017) realizou um estudo comparando o protocolo AMQP utilizando QoS 1 com o protocolo MQTT utilizando QoS 1 e 2. A Figura 3.1 demonstra os resultados que comparam as latências entre os dois protocolos. Pode-se notar que, quando as mensagens possuem tamanho de até 4096 bytes, a latência do AMQP se mantém próxima porém ainda maior que do MQTT . Quando o tamanho da mensagem aumenta, a latência do AMQP já aumenta consideravelmente em relação ao MQTT. No cenário em que as mensagens possuem 163839 bytes ou mais, o protocolo AMQP atingiu mais que o dobro da latência do MQTT, mesmo com uso da QoS 2 no MQTT.

Figura 3.1 – Latência entre os protocolos MQTT e AMQP



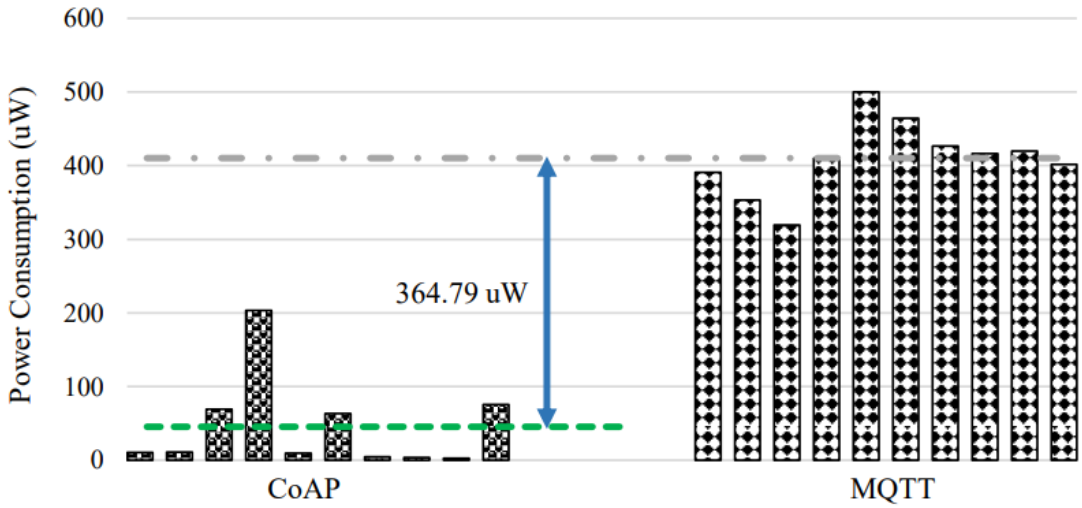

Fonte: (LINDÉN, 2017)

### 3.3.3 Consumo Energético

Outro fator importante para analisar é o consumo energético que os dispositivos IoT terão ao realizar o envio e recebimento das mensagens. Muitas vezes um sensor é instalado em uma área remota de difícil acesso e economizar energia pode reduzir bastante os custos de manutenção. Naik (2017) explica que, quanto maior a utilização de recursos, maior é o gasto energético. De forma similar aos resultados de outros parâmetros, novamente o CoAP tem o menor consumo energético dentre os três protocolos. Mesmo que os protocolos MQTT e CoAP tenham sidos desenhados para dispositivos com limitações de *hardware*, estudos comprovam que CoAP tem um consumo energético e de recursos significativamente menor que o MQTT. Jeddou et al. (2021) também ressaltam que, dentre os três protocolos abordados neste trabalho, AMQP é o que apresenta o maior consumo energético. Quando comparado a outros protocolos, ele só é mais eficiente que o protocolo HTTP. A explicação para isso é que, para conseguir desempenhar todas as suas sofisticadas funcionalidades, o protocolo AMQP precisa ter um gasto energético maior.

Conforme os resultados apresentados na Figura 3.2, o consumo médio de energia do protocolo MQTT é bem maior que o do protocolo CoAP. Safaei et al. (2017) explicam que essa diferença acontece pelo fato do protocolo MQTT possuir um *overhead* maior que o CoAP. O protocolo de transporte TCP no MQTT adiciona um mecanismo de controle de fluxo a mais no protocolo, contribuindo também para que a troca de mensagens tenha um custo energético maior. O protocolo CoAP, por outro lado, por utilizar UDP, não possui este mecanismo de fluxo de controle e também não realiza *handshake*, tornando-o um protocolo com consumo energético mais baixo que os demais.

Figura 3.2 – Comparativo do gasto energético entre MQTT e CoAP

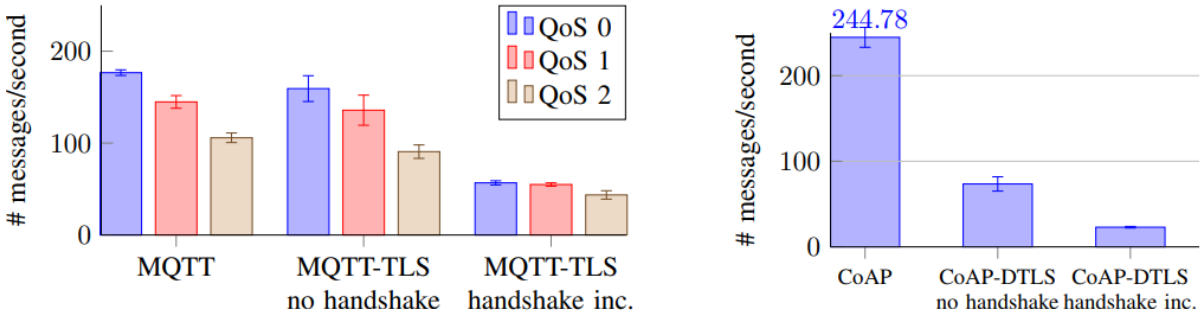



Fonte: (SAFAEI et al., 2017)

### 3.3.4 Taxa de transferência

Laaroussi e Novo (2021) realizaram um estudo comparando as taxas de transferência entre os protocolos CoAP e MQTT. Neste estudo, também abordaram as diferenças de transferência dos dois protocolos com e sem o uso de mecanismos de segurança. Os resultados mostram que as versões desprotegidas dos protocolos possuem a maior taxa de transferência de mensagens por segundo.

Figura 3.3 – Comparativo da taxa de transferência entre MQTT e CoAP




Fonte: Adaptado de (LAAROUSSI; NOVO, 2021)

Conforme exposto na Figura 3.3, o protocolo CoAP sem nenhum mecanismo de segurança obteve uma taxa de transferência maior que o protocolo MQTT. Porém, o protocolo MQTT manteve a taxa de transferência maior que o CoAP em todos os cenários utilizando mecanismos de segurança. Outra observação interessante, que pode ser extraída desses resultados, é que a performance do protocolo CoAP cai significativamente quando utilizado com o

protocolo de segurança DTLS. Já o protocolo MQTT quando utiliza TLS sem *handshake* quase não tem perda de performance. Porém, quando utilizando com *handshake* tem seu desempenho bastante impactado, mesmo assim, mantendo suas taxas de transferência ainda maiores que a do protocolo CoAP quando utilizado com DTLS com *handshake*.

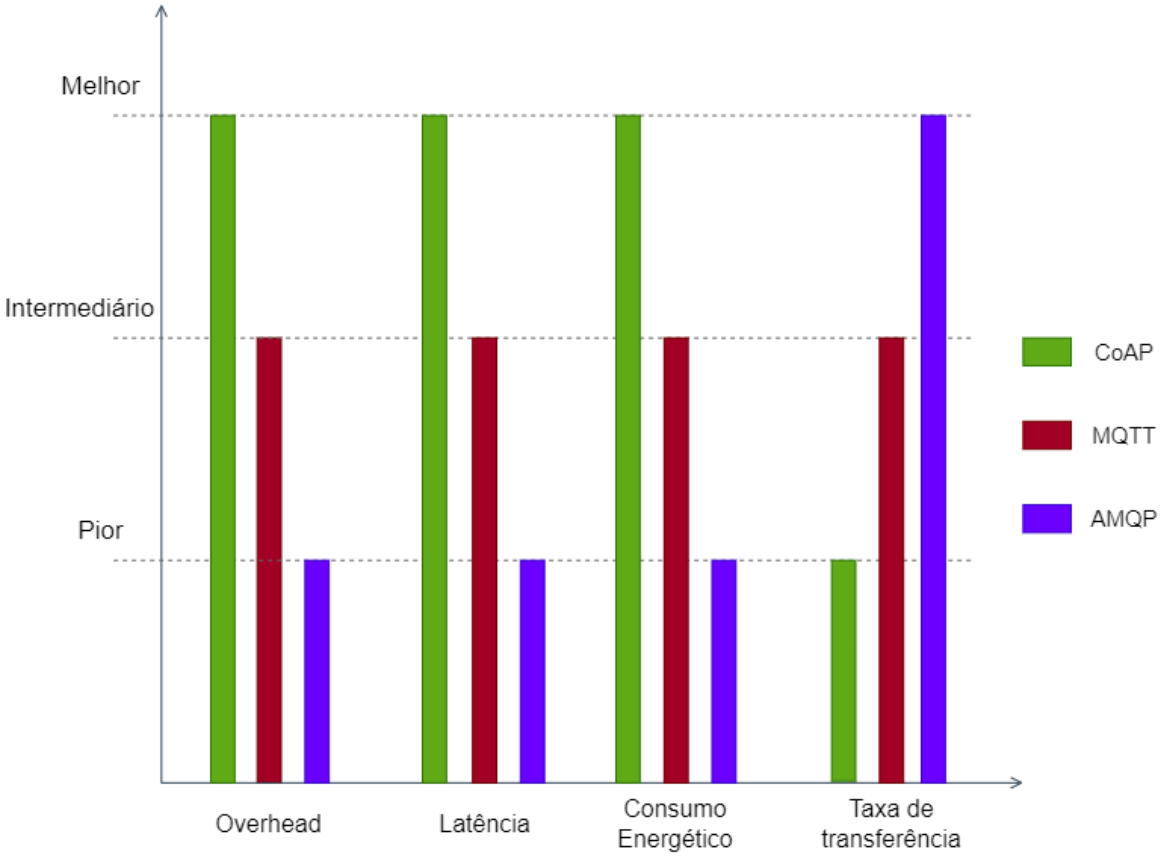
Em um experimento realizado por Moraes et al. (2019), são comparadas as taxas de transferência entre os protocolos MQTT, CoAP e AMQP. Esse experimento consistiu em configurar a qualidade de serviço de todos os protocolos para confirmarem entrega. Como resultado, o protocolo AMQP teve a maior taxa de transferência seguida pelo MQTT e, por último, o CoAP. É explicado que a razão pela qual o AMQP possui maior transferência está relacionada à retransmissão adicional das mensagens, na qual podem incluir mensagens duplicadas. Quanto ao protocolo MQTT e CoAP, o experimento demonstrou que o MQTT teve maior transferência de mensagens que o protocolo CoAP.

### 3.3.5 Resumo da análise de performance

A Figura 3.4 ilustra os quatro critérios de desempenho considerados neste estudo, que podem auxiliar na escolha de um protocolo IoT. Já taxa de transferência, quanto maior a taxa, mais mensagens são entregues ao cliente/servidor. Vale ressaltar que a taxa de transferência pode conter mensagens repetidas. É de suma importância ressaltar que o desempenho de cada protocolo sobre cada critério pode variar de acordo com diversas variáveis como qualidade da rede, tamanho da mensagem, configuração de QoS, uso de protocolo de segurança, dentre outros.

É possível notar que o protocolo CoAP tem uma boa performance na maioria dos cenários. Por utilizar o protocolo de transporte UDP, CoAP não precisa realizar esforços para transmitir seus dados. O fato de ser um protocolo mais simples em relação aos demais, proporciona um baixo custo de operacionalidade. Já o protocolo MQTT, consegue performar melhor que o AMQP pelo simples fato de ter menos funcionalidades, consequentemente tem um cabeçalho menor, o que o torna mais leve. O protocolo AMQP é mais sofisticado, possui várias funcionalidades para transmissão de mensagem, pode se conectar a mais de um servidor ao mesmo tempo, e também o mais seguro. Tudo isso dificulta que o AMQP consiga se igualar a performance dos demais protocolos.

Figura 3.4 – Tabela comparativa de performance entre os protocolos




Fonte: do autor

# 4 CONCLUSÃO

Neste trabalho, foi realizado um estudo comparativo dos principais protocolos IoT na camada de aplicação. O trabalho tem o intuito prover informações para auxiliar arquitetos na tomada de decisão quando da escolha por um protocolo para uma aplicação IoT. Foi possível notar que o protocolo CoAP, dentre os demais estudados, foi o mais leve dos protocolos, com menor custo de processamento, menos gasto energético e menor latência. Apesar disso, o protocolo AMQP mostrou ser um protocolo com mais recursos de interoperabilidade e uma maior quantidade de funcionalidades disponíveis dentre todos os protocolos, consequentemente fazendo este ser o mais pesado dos protocolos na maioria dos casos de teste. Já o protocolo MQTT mostrou estar no meio termo dentre os demais protocolos, conseguindo garantir uma latência média, e uma boa quantidade de funcionalidades. O fato do protocolo MQTT possuir 3 níveis de qualidade de serviço pode se tornar um fator decisivo na escolha de um protocolo.

É importante ressaltar que cada projeto terá suas especificidades e com isso, nem sempre a escolha de um protocolo por menor latência ou *overhead* será o melhor protocolo. Em um ambiente com uma rede estável por exemplo, um projeto IoT poderia se beneficiar mais de um protocolo que disponha de mais funcionalidades. Ou também o contrário, um projeto IoT que esteja em zona de difícil acesso ou com qualidade de rede muito baixa, talvez não precise de tantas funcionalidades e possa optar por um protocolo mais leve.

Também pode ser interessante para o arquiteto da solução saber priorizar se a aplicação poderá ou não receber mensagens duplicadas. Já que, muitas vezes, a maioria dos sistemas pode não se comportar bem com requisições ou eventos repetidos. Caso este seja o cenário, recomenda-se considerar o uso do protocolo MQTT com QoS 2, pois ambos, CoAP e AMQP, podem retransmitir uma mensagem afim de tentar garantir a entrega, gerando duplicatas.

No quesito segurança, o AMQP mostrou ser o mais completo. Além de poder implementar TLS, o protocolo ainda possibilita utilizar vários tipos de SASL. Essa diferença permite que a aplicação possa estabelecer autenticação e transporte de dados encriptados.

# REFERÊNCIAS

ABDELHALIM, A. The publish-subscribe pattern on rails: An implementation tutorial. 07 2022. Accessed: 2022-07-31. Disponível em: <https://www.toptal.com/ruby-on-rails/the-publish-subscribe-pattern-on-rails>.

AL-MASRI, E. et al. Investigating messaging protocols for the internet of things (iot). **IEEE Access**, v. 8, p. 94880–94911, 2020.

ALLIANCE, Z. Zigbee specification. 2015.

BOR, M.; VIDLER, J.; ROEDIG, U. Lora for the internet of things. In: **Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks**. USA: Junction Publishing, 2016. (EWSN '16), p. 361–366. ISBN 9780994988607.

DIAN, F. J.; VAHIDNIA, R.; RAHMATI, A. Wearables and the internet of things (iot), applications, opportunities, and challenges: A survey. **IEEE Access**, IEEE, v. 8, p. 69200–69211, 2020.

DIERKS, T.; ALLEN, C. **The TLS protocol version 1.0**. [S.l.], 1999.

ELHADI, S. et al. Comparative study of iot protocols. **Smart Application and Data Analysis for Smart Cities (SADASC'18)**, 2018.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, Irvine, 2000.

GIL, A. C. et al. **Como elaborar projetos de pesquisa**. [S.l.]: Atlas São Paulo, 2002. v. 4.

GOKHALE, P.; BHAT, O.; BHAT, S. Introduction to iot. **International Advanced Research Journal in Science, Engineering and Technology**, v. 5, n. 1, p. 41–44, 2018.

GROUP, T. O. M. What is dds? In: . [s.n.], 2022. Accessed: 2022-08-26. Disponível em: <https://www.dds-foundation.org/what-is-dds-3/>.

GUTH, J. et al. Comparison of iot platform architectures: A field study based on a reference architecture. In: **2016 Cloudification of the Internet of Things (CIoT)**. [S.l.]: IEEE, 2016. p. 1–6.

HIVEMQ, T. **Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6**. 2015. <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>. Accessed: 2022-07-09.

HOZ, J. d. et al. Ssh as an alternative to tls in iot environments using http. In: **2018 Global Internet of Things Summit (GIoTS)**. [S.l.: s.n.], 2018. p. 1–6.

ISHAQ, I. et al. Experimental evaluation of unicast and multicast coap group communication. **Sensors**, v. 16, p. 1137, 07 2016.

JEDDOU, S. et al. Power consumption prediction of iot application protocols based on linear regression. **International Journal of Artificial Intelligence and Machine Learning (IJAIML)**, IGI Global, v. 11, n. 2, p. 1–16, 2021.

KOELSCH, J. R. **Why the MQTT Protocol is So Popular**. 2022. <https://www.automationworld.com/factory/iiot/article/22080946/why-the-mqtt-protocol-is-so-popular#:~:text=With%20such%20broad%20uptake%2C%20MQTT,dominate%20the%20industrial%20networking%20space.> Accessed: 2022-07-04.

LAAROUSSI, Z.; NOVO, O. A performance analysis of the security communication in coap and mqtt. In: **2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)**. [S.l.: s.n.], 2021. p. 1–6.

LARMO, A. et al. Comparison of coap and mqtt performance over capillary radios. In: **2018 Global Internet of Things Summit (GIoTS)**. [S.l.: s.n.], 2018. p. 1–6.

LESJAK, C.; HEIN, D.; WINTER, J. Hardware-security technologies for industrial iot: Trustzone and security controller. In: **IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society**. [S.l.: s.n.], 2015. p. 002589–002595.

LIGHT, R. **What is MQTT? A practical introduction**. 2021. <https://mosquitto.org/man/mosquitto-conf-5.html>. Accessed: 2022-07-04.

LINDÉN, E. **A latency comparison of IoT protocols in MES**. 2017.

LUDOVICI, A.; MORENO, P.; CALVERAS, A. Tinycoap: A novel constrained application protocol (coap) implementation for embedding restful web services in wireless sensor networks based on tinyos. **Journal of Sensor and Actuator Networks**, v. 2, n. 2, p. 288–315, 2013. ISSN 2224-2708. Disponível em: <https://www.mdpi.com/2224-2708/2/2/288>.

MASUDUZZAMAN, M. et al. Two phase authentication and vpn based secured communication for iot home networks. In: **Safety, Security, and Reliability of Robotic Systems: Algorithms, Applications, and Technologies**. [S.l.]: CRC Press, 2020. p. 131–140.

MORAES, T. et al. Performance comparison of iot communication protocols. In: **2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)**. [S.l.: s.n.], 2019. p. 3249–3254.

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

ORACLE. O que é iot? 07 2022. Accessed: 2022-07-26. Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot>.

RABBITMQ. Amqp 0-9-1 model explained. 2007. Accessed: 2022-08-07. Disponível em: <https://www.rabbitmq.com/tutorials/amqp-concepts.html>.

SAFAEI, B. et al. Reliability side-effects in internet of things application layer protocols. In: . [S.l.: s.n.], 2017.

SILVA FELIPE FADUL DE CARVALHO, M. D. N. Aryane Barros Maciel da. Constrained application protocol(coap). In: . [s.n.], 2019. Accessed: 2022-07-17. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/coap/funcionamento.html>.

SONI, D.; MAKWANA, A. A survey on mqtt: a protocol of internet of things (iot). In: **International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)**. [S.l.: s.n.], 2017. v. 20, p. 173–177.

STANDARD, O. Oasis advanced message queuing protocol (amqp) version 1.0. In: . [s.n.], 2012. Accessed: 2022-09-14. Disponível em: <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html>.

TAKIDDEEN, N.; ZUALKERNAN, I. Smartwatches as iot edge devices: A framework and survey. In: IEEE. **2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)**. [S.l.], 2019. p. 216–222.

TANENBAUM, A. S.; STEEN, M. v. **Sistemas Distribuídos: Princípios e Paradigmas**. [S.l.]: Pearson Prentice Hall, 2007.

THANGAVEL, D. et al. Performance evaluation of mqtt and coap via a common middleware. In: . [S.l.: s.n.], 2014.

UGRENOVIC, D.; GARDASEVIC, G. Coap protocol for web-based monitoring in iot healthcare applications. In: **2015 23rd Telecommunications Forum Telfor (TELFOR)**. [S.l.: s.n.], 2015. p. 79–82.

WENDT, J. B.; POTKONJAK, M. Hardware obfuscation using puf-based logic. In: IEEE. **2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.], 2014. p. 270–271.

XU, T.; WENDT, J. B.; POTKONJAK, M. Security of iot systems: Design challenges and opportunities. In: IEEE. **2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.], 2014. p. 417–423.